APPRENDRE A PROGRAMMER EN BASIC

Claude Delannoy



sylvine bouvry

Apprendre à programmer en BASIC

S.8

DU MEME AUTEUR aux Editions Eyrolles

Apprendre à programmer en FORTRAN

Les fichiers en BASIC sur micro-ordinateur

Apprendre à programmer en BASIC

par

Claude DELANNOY

Deuxième édition
nouveau tirage



61, bd Saint-Germain, 75005 Paris
1983

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

AVANT-PROPOS

A son origine, le langage Basic, créé au Darmouth College, se voulait un langage simple pour débutants, utilisable dans n'importe quel domaine et d'un apprentissage facile et rapide. Pour rester fidèle à cet esprit, nous avons réalisé ce manuel de telle manière qu'il soit accessible à toute personne désirant aborder la programmation en Basic. Aussi ne requiert-il aucune connaissance préalable en informatique.

Un premier chapitre de présentation décrit ce qu'est un ordinateur ou un micro-ordinateur et à quoi correspond la notion de «langage».

Les chapitres 2 à 17 forment un cours complet du Basic correspondant à la norme «ANSI X 3.60 1978» qui définit le langage minimum (ou standard) que doit posséder tout ordinateur. Nous avons tenté une démarche pédagogique, très progressive dont nous souhaitons qu'elle permette une assimilation aisée et rapide. Chaque nouvelle notion est introduite à partir d'une situation concrète qui en fait sentir l'intérêt. Elle est ensuite renforcée sur des exemples d'application. A la fin de chaque chapitre, des questions et exercices permettent au lecteur de contrôler son acquis (des réponses sont fournies en fin de volume). Nous ne saurions trop conseiller de faire ces exercices, crayon en main (éventuellement «ordinateur en main»), avant de passer au chapitre suivant.

La plupart des micro-ordinateurs proposent un langage Basic dit «étendu», offrant de nombreuses possibilités supplémentaires par rapport à la norme. Les chapitres 18 à 21 décrivent les principales d'entre elles, d'ailleurs communes à bon nombre de machines. Il est probable que nombreux seront ceux qui, poussés par une curiosité naturelle, voudront en prendre rapidement connaissance.

Toutefois, nous pensons que les avantages qu'elles procurent ainsi que les nouvelles notions qu'elles comportent seront plus faciles à saisir après l'étude du Basic standard. Aussi, recommandons-nous d'être patient et d'éviter de trop «griller les étapes».

L'utilisateur ayant accès à un ordinateur ou à un micro-ordinateur sera incomparablement privilégié. Très vite, il pourra l'utiliser comme outil pédagogique. La sentence inexorable de la machine lui fournira un jugement immédiat et impartial sur son programme. C'est en pensant à lui que nous nous sommes attachés à le guider pas à pas dans l'utilisation d'un ordinateur (et plus particulièrement d'un micro-ordinateur). Dès le premier instant, il est familiarisé avec les problèmes d'erreurs et de corrections. A chaque stade, les différentes anomalies possibles sont décrites. Nous espérons ainsi lui éviter au maximum des situations de «blocage» dues à un comportement non prévu auquel il ne saurait rémédier.

Pour terminer, disons que ce manuel peut également constituer une bonne initiation pour qui souhaite ultérieurement aborder d'autres langages (Fortran, Pascal, ...).

TABLE DES MATIÈRES

I
PRÉSENTATION

Si vous lisez ces lignes avec lignes avec l'intention d'apprendre à programmer en Basic, il se peut que vous disposiez, soit d'un micro-ordinateur, soit d'un terminal relié à un ordinateur. Ce chapitre vous expose quelques notions générales concernant leur rôle, leur fonctionnement et leur utilisation.

1. L'ASPECT PHYSIQUE D'UN MICRO-ORDINATEUR OU D'UN TERMINAL

Dans les deux cas, les constituants visibles du matériel avec lequel vous travaillez se résument à :

– un clavier du même type que ceux des machines à écrire. Il vous permet de communiquer des ordres ou des informations à la machine.

– un écran (du type télévision) sur lequel viennent s'inscrire à la fois les ordres que vous avez frappés au clavier et les réponses fournies par la machine.

Dans le cas des micro-ordinateurs individuels, il est fréquent que cet écran soit celui de votre propre téléviseur. Dans le cas des terminaux, il arrive qu'il soit remplacé par une impression sur papier (comme sur une machine à écrire).

Il se peut que votre micro-ordinateur possède d'autres éléments visibles tels que :

– imprimante fournissant une trace écrite de ce qui s'affiche sur l'écran.

– lecteur de disquettes ou de cassettes assurant la conservation de certaines informations (programmes ou données) de manière permanente.

2. LES CONSTITUANTS CACHÉS

Ces constituants visibles que sont le clavier et l'écran ne représentent que les matériels vous donnant la possibilité de communiquer avec la machine. Ils jouent le même rôle que les touches ou le petit écran des calculettes (ou calculatrices de poche). En fait, dans de telles calculettes, il existe d'autres constituants qui permettent d'exécuter les opérations requises et éventuellement de «mémoriser» des résultats ou des programmes.

Dans le cas d'un micro-ordinateur, nous retrouvons ces deux éléments fondamentaux qui se nomment alors :

– mémoire centrale

– unité centrale

La mémoire centrale est destinée à contenir, à la fois des instructions de programme et des «données». (Ceci est quelque peu différent dans le cas des calculettes ou les données sont mémorisées indépendamment du programme, dans des «registres»). L'unité centrale, quant à elle, exécute les instructions du programme, et ceci à raison d'une seule à la fois. Elle ne sait reconnaître et exécuter qu'un nombre limité d'opérations élémentaires. Parmi celles-ci, on peut citer :

– les opérations arithmétiques (addition, soustraction, multiplication, division).

– comparaison d'un nombre à zéro

– branchement : il demande à l'unité centrale de continuer l'exécution du programme à partir d'un certain endroit, au lieu de passer à la suite

– branchement conditionnel : il est analogue au précédent, à cette différence près que le branchement n'a lieu que si une certaine condition est réalisée.

L'ensemble des instructions qu'une machine est capable d'exécuter porte le nom de langage machine.

Bien sûr, lorsque vous travaillez depuis un terminal, ces deux constituants fondamentaux que sont mémoire centrale et unité centrale font partie de l'ordinateur auquel vous êtes relié.

Étant donné que les notions de base que nous allons évoquer par la suite restent, pour un ordinateur, les mêmes que pour un micro-ordinateur, nous ne ferons plus cette distinction entre les deux, et nous parlerons désormais uniquement d'ordinateur.

3. NOTION DE LANGAGE ÉVOLUÉ

Si vous ne disposiez uniquement que d'un ordinateur tel que nous venons de le décrire, il vous serait peut-être difficile de l'utiliser. En effet, pour toute tâche que vous souhaiteriez lui soumettre, il vous serait nécessaire d'introduire un programme formé d'instructions en «langage machine». Ceci ressemblerait à ce que vous faites d'ailleurs sur une calculette programmable.

En réalité, vous savez que généralement il vous est possible de «parler» plus facilement à votre ordinateur, en utilisant un langage dit «évolué» tel que :

BASIC FORTRAN PL/I PASCAL . . .

Bien que différents les uns des autres, de tels langages possèdent, par rapport au langage machine, de nombreux avantages, parmi lesquels on peut citer :

– un niveau de symbolisation important les rendant beaucoup plus proches du problème de l'utilisateur que ne l'était le langage machine. (Ils vous permettent de penser à votre problème plutôt qu'à votre ordinateur).

– une indépendance vis-à-vis des machines. Alors que chaque ordinateur possède son langage machine, le même langage Basic sera utilisable sur différents matériels.

Mais, direz-vous, par quel artifice, votre ordinateur peut-il comprendre un tel langage ? En réalité, il ne le peut pas. Par contre, on peut imaginer un «dispositif» de traduction de ce langage évolué en langage machine. L'idée consiste à faire réaliser cette traduction, non pas par un matériel spécial, mais tout simplement par un programme que l'on fait s'exécuter sur le même ordinateur. Ce programme, fourni par le constructeur est, bien entendu, écrit en langage machine.

4. TRADUCTION D'UN LANGAGE ÉVOLUÉ

Il existe en fait deux méthodes pour traduire un programme langage évolué en langage machine :

– la compilation (le programme de traduction s'appelle alors compilateur)

– l'interprétation (le programme de traduction s'appelle alors interprêteur)

4.1. La compilation

Elle consiste à traduire, dans un premier temps, l'ensemble des instructions du programme en un nouveau programme d'instructions en langage machine. Si cette opération s'est déroulée normalement (c'est-à-dire si aucune erreur n'a été détectée), le programme machine est alors exécuté.

Cette technique est employée pour la plupart des langages existants, mais elle est rare dans le cas de Basic.

4.2. L'interprétation

Chaque instruction est traduite, puis immédiatement exécutée. Cette technique nécessite plus de place en mémoire que la précédente. En effet, l'interprêteur et le programme, en langage évolué doivent être présents simultanément en mémoire. En outre, une même instruction est traduite autant de fois qu'elle est exécutée, ce qui peut entraîner des temps d'exécution plus importants qu'avec un compilateur.

Par contre, cette méthode est très avantageuse pour la «mise au point» des programmes. En effet, en cas d'erreur, les instructions en langage évolué étant toujours présentes en mémoire, elles peuvent être aisément corrigées. L'exécution du programme peut alors être reprise ou continuée sans difficultés.

Basic est pratiquement toujours interprêté (particulièrement sur les micro-ordinateurs).

5. NOTION DE SYSTÈME D'EXPLOITATION

A partir du moment où vous utilisez un ou plusieurs langages évolués, il vous faut disposer d'un certain nombre de programmes tels que :

– compilateurs ou interprêteurs

– bibliothèques de fonctions (par exemple les fonctions mathématiques telles que logarithmes, exponentielle, fonctions trigonométriques, . . . qui ne font pas partie des instructions machines et qui doivent être obtenues par programme)

– programmes utilitaires réalisant, par exemple, des entrées-sorties.

Cet ensemble de programmes, plus ou moins important suivant les machines, porte le nom de «système d'exploitation». Il est conservé sur disquettes ou cassettes, du moins dans le cas d'ordinateurs possédant de tels matériels de stockage. Chaque élément est recopié en mémoire centrale lorsque le besoin s'en fait sentir. Dans ces conditions, au fil du temps, la mémoire centrale pourra contenir tantôt un compilateur, tantôt votre propre programme, tantôt un interprêteur, . . . Il est alors nécessaire que la recopie et l'enchaînement de ces différents éléments soient réalisés par un programme particulier, toujours présent en mémoire, nommé moniteur (ou superviseur).

REMARQUE :

Dans le cas de micro-ordinateurs individuels n'ayant aucun matériel de stockage, le système d'exploitation comporte peu de constituants. En effet, ceux-ci doivent se trouver en permanence en mémoire centrale. (En toute rigueur, il se trouvent dans des mémoires particulières, non destructibles, appelées ROM). Bien

qu'il ne soit plus alors nécessaire de recopier les différents constituants en mémoire, le moniteur reste indispensable pour assurer leur enchaînement.

6. LES DIFFÉRENTS NIVEAUX DE «DIALOGUE»

D'après ce que nous venons de décrire, vous voyez que vous êtes amenés à communiquer, non pas directement avec notre ordinateur, mais avec différents programmes. Même si vous n'utilisez que le langage Basic, vous aurez à travailler avec au moins trois types de programmes différents :

– le moniteur

– l'interprêteur Basic

– votre programme

Par exemple, vous demanderez au moniteur de recopier en mémoire l'interprêteur Basic. De même, vous communiquerez avec Basic pour créer un programme ou pour demander son exécution. Enfin, vous fournirez probablement des informations (on dit souvent des données) à votre programme qui, à sont tour, vous produira des résultats.

Il est très important de savoir avec quel programme vous communiquez, à un instant donné. En effet, chacun possède ses conventions propres. On voit mal comment demander à son propre programme de recopier en mémoire le compilateur Fortran... ou encore, comment fournir a un interprêteur Basic, des instructions Pascal.

7. COMMENT VOUS EXPRIMER

Quel que soit le programme avec lequel vous dialoguez, vous vous exprimez en frappant quelque chose au clavier. Mais, ce que vous tapez n'est «pris en compte» que lorsque vous le demandez, en frappant une touche particulière. Suivant les matériels, elle se nomme «RETURN» ou «CARRIAGE RETURN» ou «TRANSMIT», ... Dans

la suite de ce manuel, nous supposerons qu'il s'agit de «RETURN».

Cette façon de procéder présente un très grand intérêt en cas d'erreur de frappe. Dans ce cas, en effet, il vous est tout-à-fait loisible de corriger votre «message» avant de décider de la faire parvenir au programme concerné.

REMARQUES :

1. En général, la longueur des messages que vous pouvez transmettre est limitée à une ligne de 64 ou 80 caractères. Parfois, vous aurez droit jusqu'à 255 caractères (qui occuperont alors plus d'une ligne d'écran).

2. A chaque fois que vous pressez la touche «RETURN», le message que vous venez de composer est pris en compte, et il y a un «retour à la ligne». Autrement dit, si la machine vous fournit une réponse, elle le fait à partir du début de la ligne suivante.

EXERCICES

1. Quels sont les constituants visibles d'un micro-ordinateur ?

2. Quels sont les constituants essentiels non visibles d'un micro-ordinateur ?

3. Que contient la mémoire centrale ?

4. Qu'est-ce que le langage machine ?

5. VRAI ou FAUX ? l'unité centrale peut exécuter des instructions Basic.

6. Quelles sont les deux techniques de traduction de langage évolué ?

7. Dans quel cas trouve-t-on en même temps en mémoire le programme en langage évolué et le programme de traduction ?

8. Quels sont les trois programmes avec lesquels vous serez amenés à dialoguer si vous travaillez en Basic ?

9. A quoi sert la touche «RETURN» (ou son équivalent) ?

II

FAMILIARISATION AVEC LE MODE DIRECT

1. INTRODUCTION

Généralement, après avoir mis un ordinateur sous tension, il vous est nécessaire d'effectuer un certain nombre d'opérations pour pouvoir travailler en Basic. Celles-ci dépendent de la machine utilisée ; c'est pourquoi nous ne les décrirons pas ici. Nous supposerons donc, dans toute la suite de ce manuel, que vous êtes en mesure de travailler en Basic.

Dès le prochain chapitre, nous apprendrons à écrire des programmes dans ce langage. Auparavant, nous allons nous familiariser, sur des exemples, avec ce que l'on appelle le «mode direct». Il consiste à utiliser Basic comme une calculatrice à mémoire, en lui donnant des ordres qu'il exécute immédiatement.

Fréquemment, pour dire qu'il est prêt à recevoir un ordre, Basic écrit un petit mot tel que Ok ou READY, ou un simple caractère, par exemple > ou]. Dans ce manuel, il s'agira toujours du mot Ok. (Il pourra nous arriver de ne pas le mentionner, dans certains exemples, par souci de brièveté).

N'oubliez pas que tout ordre transmis à Basic n'est pris en compte que lorsque vous frappez la touche «RETURN» ou son équivalent.

2. PRINT

Si nous tapons :

```
PRINT 5
```

(en prenant soin de laisser au moins un blanc ou espace entre le mot PRINT et le chiffre 5)

Nous obtenons immédiatement, en réponse, sur la ligne suivante :

```
5
Ok
```

Que s'est-il passé ? Basic a, en fait, exécuté immédiatement l'ordre que nous lui avons fourni. Ici nous avons demandé d'écrire (en anglais print) la valeur 5.

Si nous tapons maintenant :

```
PRINT 4 + 5
```

nous obtenons :

```
9
Ok
```

Ici, cet ordre signifiait : écrire la valeur de la quantité 4 + 5.

Si nous souhaitons connaître la valeur de :

11,528 + 43,47

il nous suffit de taper :

```
PRINT 11.528 + 43.47
```

et nous obtenons :

```
54.998
Ok
```

Vous remarquez que, comme sur les calculatrices, les nombres décimaux s'écrivent avec un point au lieu d'une virgule.

3. LET

Si nous voulons conserver «en mémoire» une valeur ou le résultat d'un calcul, nous pouvons le faire en lui attribuant un nom. C'est ainsi que l'ordre :

```
LET A = 5
```

range la valeur 5 dans une mémoire nommée A. Par la suite, vous pourrez utiliser A pour faire de nouveaux calculs. Ainsi :

```
PRINT A
```

vous fournit :

```
5
```

Mais :

```
PRINT A + 3
```

vous donne :

```
8
```

De même, si nous frappons successivement les ordres suivants :

```
LET A = 5.9
LET B = 0.7
LET C = A – B
PRINT C
```

nous obtenons :

```
5.2
```

REMARQUE :

Dans les ordres LET que nous avons frappés, nous avons systématiquement laissé un espace de part et d'autre du signe =. Bien que ce ne soit pas indispensable, ce procédé rend les ordres un peu plus lisibles.

4. UN PEU DE CALCUL

Nous avons déjà rencontré les opérations d'addition et de soustraction. La multiplication est indiquée par *. Ainsi :

```
LET A = 5.1
LET C = A * 3
PRINT C
```

écrit :

```
15.3
```

La division, quant à elle se note / :

```
LET A = 3
LET B = 5
LET C = A/B
PRINT C
```

écrit :

```
0.6
```

L'élévation à la puissance s'écrit ^ (ou ↑ sur certaines machines). Ainsi :

```
PRINT 3 ^ 2
```

fournit :

```
9
```

De même :

```
LET A = 5
PRINT A ^ 3
```

donne :

```
125
```

5. PRÉCISION

Si nous tapons :

```
LET A = 11
PRINT A/8
```

nous obtenons :

```
1.375
```

Par contre :

```
LET A = 1
LET B = 3
LET C = A/B
PRINT C
```

nous fournit :

```
0.333333
```

De même :

```
LET C = 2/3
PRINT C
```

nous donne :

```
0.666667
```

Vous voyez que l'on obtient une valeur approchée à six décimales (arrondie à la valeur la plus proche). De même :

```
LET C = 200/3
PRINT C
```

nous donneraient :

```
66.6667
```

Autrement dit, les nombres sont représentés avec 6 chiffres significatifs. Certains Basic pourront «travailler» avec plus de chiffres. Cependant, dans tous les cas, vous devez savoir que les nombres sont nécessairement représentés avec une précision limitée, ce qui peut entraîner des erreurs dans les résultats. Dans tout ce manuel, nous utilisons un Basic à six chiffres significatifs.

Voici un autre exemple de valeur approchée ; ces instructions :

```
LET A = 1.2345678
PRINT A
```

fournissent :

```
1.23457
```

6. ÉCRIRE PLUSIEURS CHOSES DANS UN SEUL ORDRE

Si nous tapons :

```
PRINT 5,4
```

nous obtenons :

```
5              4
```

Cet ordre signifiait ici : écrire 5 puis 4. Vous ne le confondrez pas avec :

```
PRINT 5.4
```

qui aurait produit :

```
5.4
```

De même :

```
LET A = 4.5
LET B = 3.2
PRINT A, B + 3
```

donne :

```
4.5            6.2
```

Enfin :

```
LET A = 3
LET B = A/2
PRINT A, A/2, B
```

fournit :

```
3              1.5            1.5
```

7. EN CAS D'ERREUR

Que se passe-t-il lorsque vous tapez quelque chose d'erroné ? Il faut distinguer deux cas.

Premier cas

Vous ne vous êtes pas aperçu de votre erreur avant d'avoir frappé la touche «return». Si l'ordre à un sens, il sera exécuté. (Basic ignore, bien sûr, que vous vouliez dire autre chose). Dans le cas contraire, Basic vous répondra par ce que l'on appelle un diagnostic d'erreur. En voici un exemple :

```
PRIT 5 + 3
Syntax error
```

La réponse de Basic signifie : erreur de syntaxe. Autrement dit, votre ordre ne respecte pas les règles d'écriture Basic.

Deuxième cas

Vous découvrez une erreur dans ce que vous tapez alors que vous n'avez pas encore pressé la touche «return». Vous pouvez alors :

– annuler totalement ce que vous aviez commencé à écrire. Ceci se fait souvent en pressant la touche C tandis que l'on maintient enfoncée la touche «Control» (marquée souvent CNTL ou CNTRL), mais cela peut être différent sur votre propre système.

Voici un exemple :

```
LET A = C  CNTL + C
Ok
```

– annuler seulement le ou les derniers caractères frappés. Vous utilisez pour cela le symbole ← (ou une touche notée «backspace») qui annule le dernier caractère tapé. Ainsi :

```
LA ← ET A = B
```

est bien interprêté comme :

```
LET A = B
```

Plusieurs «backspace» annulent plusieurs caractères. Ainsi :

```
LAT← ←ET C = D
```

est interprété comme :

```
LET C = D
```

De même :

```
PRNT ← ← INT A – ← + 3
```

est équivalent à :

```
PRINT A + 3
```

Là encore, la touche (ou la combinaison de touches) à utiliser peut être différente sur votre propre système. De même, il est fréquent, dans le cas des machines utilisant un écran, que le «backspace» permette un véritable retour en arrière avec effacement du caractère précédemment affiché et remplacement par celui nouvellement entré. Dans ce cas, rien ne montre, après coup, qu'il y a eu une correction.

EXERCICES

1. Donnez un ordre permettant d'écrire la valeur de :

1,39 + 4,734

2. Donnez un ordre permettant d'écrire la valeur de :

1,39 × 4,734

3. Donnez un ordre permettant d'écrire la valeur de :

$\frac{41}{11}$

4. Qu'obtiendra-t-on avec :

PRINT 6

5. même question pour :

PRINT 6 – 5

6. même question pour :

PRINT 6/6

7. même question pour :

PRINT 5 * 4

8. Donnez l'ordre permettant de ranger la valeur 5 dans la mémoire nommée C.

9. Donnez les ordres permettant de ranger la valeur 5 dans A et d'en calculer le carré dans B.

10. Qu'obtiendra-t-on avec :

```
LET A = 3
PRINT A
```

11. même question pour :

```
LET B = 5
PRINT B + 3
```

12. même question pour :

```
PRINT 9/6
```

13. même question pour :

```
LET A = 4
LET B = 2.5
PRINT A, B + A
```

14. même question pour :

```
LET A = 5
PRINT A/2, 2 * A
```

III

PREMIERS PAS EN PROGRAMMATION BASIC

1. INTRODUCTION

Supposons que l'on souhaite calculer l'aire d'un cercle de rayon R = 5. Nous pouvons taper :

```
LET R = 5
LET S = 3.14159 * R ^ 2
PRINT S
```

ce qui nous fournit la réponse :

```
78.5397
```

Si maintenant, l'on souhaite calculer l'aire d'un cercle de rayon R = 9, il nous semble nécessaire de taper à nouveau des ordres semblables aux précédents :

```
LET R = 9
LET S = 3.14159 * R ^2
PRINT S
```

En fait, il est possible, comme on le fait sur des calculatrices programmables «d'enregistrer» les ordres ci-dessus sous la forme d'un programme. On travaille alors en deux étapes : dans un premier temps, on crée le «programme», c'est-à-dire la suite des instructions qui le constituent. Dans un deuxième temps, on demande «d'exécuter», c'est-à-dire de faire fonctionner le programme ainsi créé.

2. COMMENT CRÉER UN PROGRAMME

Si nous tapons :

```
LET R = 5
```

Basic exécute directement cet ordre, sans en garder aucune trace.

Pour «l'enregistrer», il nous suffit de le taper en le faisant précéder d'un numéro appelé «numéro de ligne». Par exemple :

```
10 LET R = 5
```

Dans ce cas, l'ordre LET R = 5 n'est pas exécuté. C'est l'instruction 10 LET R = 5 (avec son numéro de ligne) qui est mémorisée par Basic. De la même manière, nous écrirons :

```
20 LET S = 3.14159 * R ^2
30 PRINT S
```

3. COMMENT FAIRE FONCTIONNER UN PROGRAMME

Si maintenant, nous désirons faire fonctionner (on dit souvent «exécuter») ce programme constitué ici de trois instructions, il nous suffit de taper :

```
RUN
```

Les instructions sont alors exécutées en suivant l'ordre croissant des numéros de ligne (ce qui, ici, correspond à l'ordre dans lequel nous les avons frappées). Au bout du compte, nous obtenons le résultat :

```
78.5397
```

REMARQUE :

Bien que cela soit peu fréquent, il est possible que votre Basic vous demande de placer une instruction END (en Anglais : FIN)

à la fin de votre programme. Dans ce cas, il vous aurait fallu enregistrer quatre instructions au lieu de trois :

```
10 LET R = 5
20 LET S = 3.14159 * R ^2
30 PRINT S
40 END
```

A ce stade, nous pourrions à nouveau taper RUN pour demander une autre exécution de notre programme. En effet, ce dernier est toujours présent dans la mémoire. Néanmoins, ici, cela présente peu d'intérêt puisque nous obtiendrions le même résultat que précédemment.

Vous devez commencer à imaginer qu'un programme, quel qu'il soit, prendra de l'intérêt à partir du moment où il sera capable de s'exécuter plusieurs fois en portant sur des «données» différentes. C'est ainsi qu'ici, il serait pratique de pouvoir calculer l'aire d'un cercle de rayon quelconque. Nous allons voir une première manière de résoudre ce problème en modifiant l'instruction qui attribue une valeur au rayon. Il nous faut, pour cela, apprendre à modifier un programme déjà enregistré.

4. COMMENT MODIFIER UN PROGRAMME

4.1. Modification d'une ligne existante

Dans notre exemple, nous voulons modifier la valeur de R. Pour cela nous pouvons donc modifier l'instruction portant le numéro de ligne 10. Nous le faisons en tapant une nouvelle instruction portant ce même numéro. Par exemple :

```
10 LET R = 9
```

L'ancienne instruction 10 LET R = 5 sera purement et simplement remplacée par cette dernière.

4.2. Liste d'un programme

Ici, l'exemple est très simple et nous sommes certains que le programme obtenu par cette modification est correct. En pratique,

il sera fréquent qu'après quelques modifications, vous ayez envie de vérifier que le programme enregistré est bien celui escompté. Pour cela, vous pouvez en obtenir la «liste» en tapant :

```
LIST
```

Dans notre exemple, nous obtiendrions :

```
10 LET R = 9
20 LET S = 3.14159 * R ^2
30 PRINT S
Ok
```

Exécutons alors ce programme :

```
RUN
254.469
```

4.3. Ajout d'une ligne à un programme

Supposons que nous voulions modifier notre programme pour qu'il écrive la valeur de R avant celle de S. Pour cela, nous avons besoin d'une instruction PRINT R entre les lignes 20 et 30. Nous allons donc taper cette instruction avec un numéro de ligne compris entre 20 et 30, par exemple :

```
25 PRINT R
```

Vous voyez tout l'intérêt qu'il y a à utiliser des numéros de ligne non consécutifs lorsque vous entrez votre programme. Cela permet, en effet, d'ajouter facilement de nouvelles instructions (si ce n'était pas le cas, vous risqueriez d'avoir à modifier une ou plusieurs instructions pour en ajouter une nouvelle).

Demandons une liste de notre programme ainsi modifié :

```
LIST
10 LET R = 9
20 LET S = 3.14159 * R ^2
25 PRINT R
30 PRINT S
```

Son exécution nous donne :

```
RUN
9
254.469
```

4.4. Effacement d'une ligne d'un programme

Si nous désirons enlever (ou «effacer») la ligne 25 de notre exemple, nous tapons ce numéro de ligne tout seul (autrement dit nous frappons, la touche «return» après avoir tapé 25). On peut considérer que l'on a ainsi «remplacé» l'ancienne ligne par une nouvelle ne comportant rien (on dit souvent qu'elle est «vide»). Effaçons donc notre ligne et vérifions le résultat obtenu en faisant une liste :

```
25
LIST
10 LET R = 9
20 LET S = 3.14159 * R ^2
30 PRINT S
Ok
```

5. POUR NE PAS MODIFIER SANS CESSE SON PROGRAMME. L'INSTRUCTION INPUT

Nous avons vu comment utiliser notre programme pour calculer les surfaces des cercles de rayon 5 ou 9. Nous pourrions continuer de même pour d'autres valeurs de R, en tapant, par exemple :

```
10 LET R = 3
RUN
28.2743
Ok
10 LET R = 2.5
RUN
19.6349
Ok
. . . etc . . .
```

Cependant, cette façon de procéder est peu pratique car elle oblige la personne qui utilise le programme :

– à connaître les numéros des instructions à modifier.

– à connaître le langage Basic.

– à frapper des modifications qui peuvent être longues.

En fait, l'instruction INPUT va nous fournir une solution beaucoup plus simple. En effet, elle permet d'introduire des valeurs lors de l'exécution du programme et non plus lors de son écriture comme c'était le cas avec l'instruction LET.

L'instruction :

```
10 LET R = 9
```

attribuait, lorsque Basic l'exécutait, la valeur 9 à R. Remplaçons la par :

```
INPUT R
```

Nous constatons, en tapant LIST que notre programme se présente maintenant ainsi :

```
10 INPUT R
20 LET S = 3.14159 * R ^2
30 PRINT S
```

Demandons son exécution :

```
RUN
?
```

Ce point d'interrogation signifie que, pendant l'exécution de votre programme, Basic a rencontré une instruction INPUT (en Anglais «entrer»). Il attend alors que vous lui fournissiez une valeur pour R.

Vous le faites en tapant au clavier une valeur de votre choix (suivie, bien sur de «return»). Supposons que vous ayez répondu :

```
? 5.5
```

vous obtenez alors le résultat :

```
95.0331
```

Ainsi, avec l'instruction INPUT, vous communiquez des valeurs lors de l'exécution de votre programme. Alors que l'instruction LET vous obligeait à décider de la valeur attribuée à R, dès l'écriture du programme, l'instruction INPUT vous permet de ne le faire qu'au moment de l'exécution. Notamment, et c'est là son principal intérêt, vous pouvez exécuter à plusieurs reprises le même programme

en lui fournissant à chaque fois une réponse différente. Faisons le avec notre exemple :

```
RUN
? 3
28.2743
Ok
RUN
? 9
254.469
Ok
RUN
? 10
314.159
Ok
```

REMARQUES :

1. on dit souvent que l'instruction INPUT sert à «entrer» une valeur au clavier lors de l'exécution du programme. On dit aussi qu'elle «lit» une valeur au clavier.

2. nous avons déjà parlé des différents niveaux possibles de dialogue que vous pouvez rencontrer. Lorsque vous tapez des instructions ou des ordres tels que RUN ou LIST, vous vous adressez à Basic. Par contre, quand vous répondez à un point d'interrogation, vous vous adressez à votre propre programme (en toute rigueur, cependant, par l'intermédiaire de Basic).

6. QUAND VOUS FAITES DES ERREURS DANS VOTRE PROGRAMME

Lorsqu'en mode direct, vous frappez une instruction incorrecte, Basic vous le mentionne immédiatement. Par contre, si vous enregistrez une instruction erronée, Basic ne vous le signalera pas. En effet, à ce moment là, il se contente de mémoriser les instructions que vous lui fournissez, sans les interpréter. Ce n'est que lorsque vous demanderez l'exécution du programme correspondant qu'il découvrira l'erreur. Il vous fournira alors un message.

Par exemple, si vous tapez le programme suivant :

```
10 INPUT R
20 LAT S = 3.14159 * R ^2
30 PRINT S
```

Basic l'enregistre purement et simplement, sans vérifier que les instructions en sont correctes. Par contre, lorsque vous voudrez l'exécuter vous obtiendrez :

```
RUN
? 2.5
Syntax error in 20
```

En effet, l'instruction 10 a bien demandé une valeur pour R (ici, nous avons répondu 2.5). Puis l'exécution se poursuit en ligne 20. Celle-ci n'étant pas correcte, Basic écrit alors un «message d'erreur» signifiant ici :

```
Erreur de syntaxe en 20
```

Ce message pourra être différent sur votre système mais, dans tous les cas, il précisera le numéro de la ligne en cause.

Un bref examen de notre programme nous amène à découvrir l'erreur en question. La correction se fait en frappant à nouveau la ligne 20 :

```
20 LET S = 3.14159 * R ^2
RUN
? 2.5
19.6349
Ok
```

Pour vous habituer aux messages d'erreur, nous vous conseillons d'essayer l'exemple précédent sur votre machine. Bien sûr, d'autres types d'erreurs existent ; nous en parlerons progressivement tout au long de ce manuel.

7. QUAND BASIC VOUS PLACE EN MODE ÉDITEUR MALGRÉ VOUS

Sur certaines machines, il se trouve qu'après avoir signalé l'erreur en ligne 20, Basic vous demande d'en effectuer la correction.

Pour cela, il vous place automatiquement dans ce que l'on appelle le «mode éditeur». C'est un mode de travail qui vous permet de corriger une ligne, sans devoir la retaper en entier. La difficulté, pour le débutant, réside alors dans le fait qu'il ne connaît généralement pas la façon de travailler dans ce mode. En effet, il ne suffit plus de taper simplement une instruction car certaines touches servent alors à dire qu'on modifie, supprime ou ajoute un ou plusieurs caractères.

Comment reconnaître que votre Basic vous place automatiquement dans ce mode ? A ce qu'il vous indique alors le numéro de ligne à corriger avant d'en attendre les modifications, comme ceci :

```
Syntax error in 20
20
```

Ici, Basic, en mode éditeur, attend les corrections pour la ligne 20.

Comment effectuer la correction dans ce mode ? Vous pouvez bien sûr apprendre à utiliser l'éditeur. Ceci vous sera d'ailleurs utile, d'une manière générale pour modifier des programmes. Cependant, il n'est pas indispensable de vous y entraîner dès maintenant. Il vous suffit, pour l'instant, de savoir comment répondre que l'on ne fait aucune correction. La plupart du temps, cela se fait en frappant la touche «return», après que Basic vous ait fournit le numéro de la ligne à corriger. Ainsi, dans notre exemple, nous obtiendrions :

```
Syntax error in 20
20                          (ici, nous avons pressé «return»)
20 LAT S = 3.14159 * R ^2
Ok
```

Après que vous ayez pressé «return», le mode éditeur vous écrit la ligne, après corrections (ici, nous n'en avons fait aucune), puis revient dans le mode habituel comme le montre le mot Ok. Nous pouvons alors corriger réellement notre ligne 20, comme nous le faisions auparavant, c'est-à-dire en la tapant à nouveau.

8. QUAND VOUS FAITES DES ERREURS EN RÉPONDANT A INPUT

Reprenons le programme précédent :

```
10 INPUT R
20 LET S = 3.14159 * R ^2
30 PRINT S
```

et lançons son exécution en répondant comme suit au point d'interrogation :

```
RUN
? RAYON
Redo from start
?
```

Votre réponse n'étant pas un nombre, il est impossible à Basic d'attribuer une valeur à R. Il vous signale cette erreur par un message du type :

```
Redo from start
```

qui signifie : Recommencez depuis le début (sous-entendu le début de la réponse à INPUT).

Vous remarquez que Basic vous fournit à nouveau un point d'interrogation pour vous montrer qu'il attend une autre réponse.

Comme précédemment, nous vous conseillons d'essayer ce petit exemple sur votre machine afin de connaître le message exact correspondant.

EXERCICES

1. Quelle différence y-a-t-il entre ces deux ordres :

```
PRINT 5 + 3
20 PRINT 5 + 3
```

2. Comment peut-on effacer une ligne d'un programme ?

3. Que faites-vous pour corriger l'instruction erronée du programme suivant :

```
10 LET A = 2
20 LIT  B = 2 * A
30 PRINT B
```

4. Modifiez le programme ci-dessus pour qu'il écrive à la fois la valeur de A et celle de B.

5. Que fait ce programme ;

```
10 LET C = 1
20 LET C = 2
30 PRINT C
```

6. A quoi sert l'instruction INPUT ?

7. Que signifie, dans un programme, «lire une valeur au clavier» ?

8. VRAI ou FAUX ? L'instruction :

```
50 INPUT A
```

a) attribue une valeur à A lors de l'écriture du programme.

b) vous permet de ne pas connaître la valeur de A quand vous écrivez votre programme.

9. Cette instruction est-elle correcte ?

```
100 INPUT X = 5
```

10. Comment reconnaître que votre programme attend que vous lui fournissiez une réponse ?

11. Modifiez le programme de l'exercice 3 pour que la valeur de A soit fournie au clavier, lors de l'exécution.

12. Qu'obtiendrez vous si vous frappez :

```
10 INPOUT A
```

13. Qu'obtiendrez vous si vous tapez :

```
10 INPOUT A
20 LET R = A + 2
30 PRINT R
RUN
```

14. A quoi sert le mode éditeur ?

15. Que faites vous si Basic, trouvant une erreur dans votre programme, vous place automatiquement en mode éditeur.

16. Écrire un programme qui demande qu'on lui fournisse un nombre au clavier et qui en écrit son double.

17. Même question avec le cube du nombre.

18. VRAI ou FAUX ? Lors de son exécution, l'instruction :

```
50 INPUT X
```

a) demande à l'utilisateur de lui fournir au clavier une valeur pour X.
b) lit au clavier une valeur pour X.
c) attend une réponse
d) prévient l'utilisateur qu'il doit taper quelque chose par un point d'interrogation.

IV

POUR TRAVAILLER AVEC PLUSIEURS PROGRAMMES

Dans le chapitre précédent, vous avez vu comment créer ou modifier un programme. Par contre, vous avez certainement constaté que celui-ci disparaissait lors de la mise «hors tension» de votre ordinateur. Nous allons voir ici comment conserver et retrouver un programme. Auparavant, nous allons essayer de classer les différents ordres que nous pouvons donner à Basic.

1. COMMANDES ET INSTRUCTIONS

Nous avons déjà rencontré des instructions du langage Basic, telles que :

```
PRINT B
INPUT R
LET C = A + 3
```

Si nous tapons ainsi ces instructions, elles sont immédiatement exécutées. C'est ce que nous avons appelé le «mode direct». Par contre, nous pouvons également les enregistrer en les faisant précéder d'un numéro de ligne.

D'autre part, nous avons rencontré des ordres tels que LIST ou RUN. Ceux-ci ne sont pas, à proprement parler, des instructions du langage Basic. En général, vous ne pourrez pas les faire figurer dans un programme, en les faisant précéder d'un numéro de ligne. Ces

ordres qui ne sont pas des instructions et qui sont directement exécutés par Basic sont appelés des «Commandes».

En résumé, vous pouvez travailler avec Basic d'au moins trois manières différentes :

1. exécuter des instructions en mode direct.
2. créer (ou modifier) un programme en frappant des instructions précédées d'un numéro de ligne.
3. taper des commandes (LIST, RUN, ...)

Peut-être vous souvenez-vous d'une remarque faite dans le chapitre précédent à propos du mode «éditeur». Si vous le possédez sur votre machine, il vous fournit une quatrième manière de travailler (Il possède alors des commandes qui lui sont propres).

Vous allez maintenant apprendre à utiliser de nouvelles commandes pour conserver et retrouver vos programmes.

2. LA COMMANDE NEW

Si nous tapons une instruction précédée d'un numéro de ligne, elle vient :

– s'ajouter au programme déjà enregistré si son numéro n'y figure pas encore.

– remplacer une instruction existante si son numéro figure déjà dans le programme.

Dans ces conditions, vous voyez que vous ne pouvez travailler qu'avec un seul programme à la fois. Si vous souhaitez en créer un nouveau, il est nécessaire «d'effacer» l'ancien de la mémoire. Ceci se fait par la commande «NEW» (en anglais : nouveau)

Voici un exemple :

```
LIST
10 INPUT R
20 LET S=3.14159*R^2
30 PRINT S
Ok
NEW
Ok
LIST
Ok
```

L'ancien programme dont nous avions demandé la liste a été effacé par la commande NEW, ce qui explique qu'aucune liste n'apparaisse la seconde fois.

REMARQUE :

Vous vous demandez peut-être comment nous avons pu nous passer de cette commande jusqu'ici. C'est simplement parce que chaque mise «hors tension» de votre machine provoque l'effacement du contenu de la mémoire, et donc de tout programme s'y trouvant.

3. POUR CONSERVER UN PROGRAMME

Dès que vos programmes dépasseront la dizaine d'instructions, il est probable que vous regretterez de voir une commande NEW faire disparaître ce qui vous aura demandé quelque temps de travail. Vous aurez alors envie de conserver une «copie» de certains de vos programmes. En fait, ceci ne sera réalisable que si votre machine dispose au moins d'un lecteur de cassettes ou de disquettes. Dans ce cas, vous pourrez alors effectuer des stockages permanents de vos programmes sur l'un de ces matériels.

Généralement la commande recopiant (on dit souvent sauvegardant) un programme dans une «mémoire permanente» comporte le mot SAVE (en anglais «sauvegarder»), suivi du nom que l'on souhaite attribuer à son programme. Voici un exemple :

```
LIST
10 INPUT R
20 LET S=3.14159*R^2
30 PRINT S
Ok
SAVE "CERCLE"
Ok
```

Le programme (que nous avons ici préalablement listé) sera conservé sous le nom CERCLE.

Dans le langage Basic proprement dit, un grand nombre d'instructions sont communes à toutes les machines. Par contre, beaucoup de commandes dépendent du système sur lequel vous travaillez.

C'est le cas de cette commande de sauvegarde dont il se peut qu'elle ne fonctionne pas sur votre machine. Si tel est le cas, il vous faudra consulter la documentation fournie par le constructeur.

4. POUR RETROUVER UN PROGRAMME

La commande LOAD (en Anglais : charger) recopie en mémoire un programme conservé en mémoire permanente. En voici un exemple :

```
LIST
10 LET C = 3
20 PRINT C
30 PRINT S
Ok
LOAD "CERCLE"
Ok
LIST
10 INPUT R
20 LET S=3.14159*R^2
30 PRINT S
Ok
```

La commande LOAD «cercle» a recopié en mémoire le programme nommé CERCLE. Le résultat des deux commandes LIST montre que la recopie de CERCLE a entraîné la disparition complète du programme se trouvant préalablement en mémoire (tout se passe comme si vous aviez tapé NEW avant LOAD).

Là encore, comme nous l'avons déjà remarqué pour SAVE, cette commande peut s'exprimer différemment sur certaines machines.

5. FAIRE DES REMARQUES DANS VOS PROGRAMMES

Dès que vous manipulez plusieurs programmes, vous risquez rapidement de perdre de vue le fonctionnement de chacun d'entre

eux. De même, vous aurez généralement beaucoup de difficultés à comprendre un programme écrit depuis plusieurs mois. Le problème sera encore plus ardu s'il s'agit d'effectuer des modifications dans un programme existant. Dans ces conditions, il est indispensable d'accompagner vos programmes de «commentaires» appropriés. Basic vous offre la possibilité d'en placer directement à l'intérieur même du programme. Ceci se fait par une instruction particulière REM (comme REMark, c'est-à-dire remarque).

A titre d'exemple, introduisons quelques instructions REM dans notre programme CERCLE.

```
LOAD "CERCLE"
Ok
LIST
10 INPUT R
20 LET S=3.14159*R^2
30 PRINT S
Ok
5  REM CALCUL DE LA SURFACE D'UN CERCLE
15 REM CALCUL DE LA SURFACE DANS S
LIST
5  REM CALCUL DE LA SURFACE D'UN CERCLE
10 INPUT R
15 REM CALCUL DE LA SURFACE DANS S
20 LET S=3.14159*R^2
30 PRINT S
Ok
```

Nous avons ainsi placé deux instructions REM. Celles-ci apparaissent bien sûr lorsque nous listons notre programme. Par contre, lorsque nous l'exécutons, elles sont sans effet. Plus précisément, lorsque Basic rencontre une telle instruction, il se contente de passer à la suivante. C'est ainsi que l'exécution du programme CERCLE assorti de ses remarques se déroulerait exactement comme auparavant, en fournissant les mêmes résultats.

```
RUN
? 2.5
 19.6349
Ok
```

6. POUR Y VOIR PLUS CLAIR - PRINT "TEXTE"

6.1. Pour savoir ce que votre programme attend

Quand nous exécutons le programme précédent (avec ou sans remarques), il nous faut savoir que lorsqu'un point d'interrogation apparaît, Basic attend une valeur du rayon. Dans un cas aussi simple, on peut supposer qu'aucune confusion n'est possible. Par contre, dès que le programme requiert plusieurs informations au clavier (ou dès qu'il est utilisé par une personne différente de son auteur), il peut être judicieux de faire préciser ce que l'on attend.

Ajoutons à notre programme CERCLE, l'instruction :

```
6 PRINT "DONNEZ LE RAYON"
```

et demandons en l'exécution après avoir effectué une liste :

```
LIST
6  PRINT "DONNEZ LE RAYON"
10 INPUT R
20 LET S=3.14159*R^2
30 PRINT S
Ok
RUN
DONNEZ LE RAYON
? 2.5
 19.6349
Ok
```

Vous voyez que l'instruction 6 a fait imprimer ce que nous avions placé entre guillemets. On peut dire que l'on a imprimé tous les «caractères» placés entre ces guillemets, y compris les blancs (ou espaces). Nous préciserons un peu plus tard cette nouvelle notion de «chaîne de caractères» qui se distingue de celle de nombre. Pour l'instant, il vous suffit de retenir qu'un texte placé entre guillemets dans une instruction PRINT est simplement imprimé tel quel lors de l'exécution.

Vous remarquez que le programme suivant :

```
50 LET A = 3
60 PRINT A
```

fournit à l'exécution :

```
3
```

tandis que :

```
50 LET A = 3
60 PRINT "A"
```

fournirait :

```
A
```

puisque, dans ce dernier cas, on a demandé d'écrire le texte entre guillemets, c'est-à-dire la lettre A.

De même, si :

```
PRINT 5 + 3
```

fournit, en réponse :

```
8
```

```
PRINT "5 + 3"
```

produit :

```
5 + 3
```

Dans ces deux derniers exemples, nous avons utilisé le mode direct. Nous n'avons pas eu à enregistrer un programme, puis à l'exécuter pour obtenir un résultat. Nous vous conseillons d'utiliser fréquemment ce mode pour vérifier rapidement le résultat d'une instruction sur laquelle vous avez un doute.

6.2. Pour mieux reconnaître vos résultats

Dans le même ordre d'idée, il vous est possible de préciser, à l'aide d'un texte approprié, la nature des résultats fournis par votre programme. Par exemple, ajoutons au programme CERCLE, l'instruction :

```
25 PRINT "SURFACE ="
```

voici le nouveau programme ainsi obtenu et son exécution :

```
LIST
6  PRINT "DONNEZ LE RAYON"
10 INPUT R
20 LET S=3.14159*R^2
25 PRINT "SURFACE ="
30 PRINT S
Ok
RUN
DONNEZ LE RAYON
? 2.5
SURFACE =
 19.6349
Ok
```

Nous aurions pu demander d'écrire le texte SURFACE = et la valeur de S dans la même instruction PRINT, en remplaçant simplement l'instruction :

```
30 PRINT S
```

par :

```
30 PRINT "SURFACE =",S
```

nous obtiendrions alors :

```
LIST
6  PRINT "DONNEZ LE RAYON"
10 INPUT R
20 LET S=3.14159*R^2
30 PRINT "SURFACE =",S
Ok
RUN
DONNEZ LE RAYON
? 2.5
SURFACE =       19.6349
Ok
```

Là encore, l'incorporation de texte dans les résultats sera d'autant plus appréciable que ceux-ci seront nombreux et donc difficiles à identifier par l'utilisateur.

EXERCICES

1. Dites si les ordres suivants sont des commandes ou des instructions.

```
LIST
PRINT A
SAVE "PROG1"
LET A = B + C
INPUT R
LOAD "PROG2"
NEW
RUN
```

2. A quoi sert la commande NEW ?

3. Qu'obtiendra-t-on si l'on tape successivement :

```
NEW
10 LET A = 3
NEW
20 LET B = A + 4
LIST
```

4. A quoi sert la commande SAVE ?

5. A quoi sert la commande LOAD ?

6. A quoi sert l'instruction REM ?

7. Quand voit-t'on apparaître le contenu d'une instruction REM ?

8. Que fait Basic quand il rencontre une instruction REM lors de l'exécution d'un programme ?

9. Quelle différence y aura-t-il dans les résultats fournis par les deux programmes suivants :

```
100 REM CALCUL D'UN CARRÉ
110 REM ENTRÉE DE N
120   INPUT N
130 REM CALCUL DE SON CARRÉ
140   LET C = N ^2
150 REM ECRITURE DU RÉSULTAT
160   PRINT C
```

```
100 INPUT N
110 LET C = N ^2
120 PRINT C
```

10. Quels résultats fournira ce programme :

```
100 LET A = 5
110 PRINT "A =" , A
```

11. Même question avec :

```
100 PRINT "DONNEZ MOI UN NOMBRE"
110 INPUT X
120 PRINT "VOICI SON CUBE :" , X ^ 3
```

si vous répondez 5

12. Ce programme est-il correct ?

```
100 PRINT "DONNEZ LA VALEUR DE S"
110 INPUT R
120 LET S = 3.14159 * R ^2
130 PRINT "VOICI LE RAYON =" , S
```

13. Que fournira l'exécution de ce programme :

```
10 LET D = 4
20 LET S = 3.14159 * D ^2/4
30 PRINT "S"
```

V
L'INSTRUCTION LET

1. NOTION DE VARIABLE

1.1. Qu'est-ce qu'une variable ?

Vous avez vu qu'une instruction telle que :

LET A = 4

avait pour effet de ranger la valeur 4 dans une mémoire nommée A.

Nous avions employé ce terme «mémoire» car il permettait une certaine analogie avec les calculatrices programmables. Cependant, en informatique, on emploie généralement le mot «VARIABLE». Une variable est donc, en quelque sorte, un nom que l'on attribue à un emplacement mémoire.

1.2. Évolution du contenu d'une variable

La valeur contenue dans une variable peut évoluer au cours du déroulement du programme. Exécutons, par exemple, ces instructions :

```
10 LET A=3
20 PRINT A
30 LET A=5
40 PRINT A
Ok
RUN
 3
 5
Ok
```

La variable A contenait la valeur 3 après exécution de l'instruction numéro 10. Il en va de même après celle de numéro 20. Par contre, l'instruction 30 lui attribue la valeur 5.

De même, exécutons ce programme :

```
10 LET A=3
20 LET B=5
30 PRINT A
40 LET A=B+4
50 PRINT A
Ok
RUN
 3
 9
Ok
```

Pour y suivre l'évolution du contenu des variables, on peut faire le tableau suivant précisant leur valeur après exécution de chacune des instructions :

Instruction	A	B
10 LET A = 3	3	–
20 LET B = 5	3	5
30 PRINT A	3	5
40 LET A = B + 4	9	5
50 PRINT A	9	5

Note :

La présence d'un tiret (–) dans le tableau signifie que la variable correspondante n'a pas encore reçu de valeur.

1.3. Utilisation d'une variable

Une variable peut être utilisée autant de fois que vous le désirez dans une instruction LET ou PRINT, sans que cela modifie sa valeur. Ainsi, dans l'exemple ci-dessus, vous voyez que :

```
30 PRINT A
```

n'a rien changé à la valeur de A. On y trouve la même valeur (3) que ce soit avant ou après exécution de cette instruction.

De même :

```
40 LET A = B + 4
```

qui utilise la valeur de B (5 à ce moment là) la laisse inchangée.

Par contre, à chaque fois qu'une instruction LET attribue une valeur à une variable, l'ancien contenu est définitivement perdu. Ainsi l'instruction LET A = B + 4 vient remplacer l'ancienne valeur de A (ici 3) par la valeur actuelle de B + 4 (ici 9). La valeur 3 a donc complètement disparu.

1.4. Attention aux mathématiques

Le terme de variable en informatique n'a pas le même sens qu'en mathématiques où il s'agit plutôt d'une lettre désignant une valeur quelconque (ou un ensemble de valeurs). Ainsi, en mathématiques, dans :

$$f(x) = x^2 + 3 \qquad x \in R$$

x représente un nombre réel quelconque.

En outre, la notion de temps n'y intervient pas. Par contre, en informatique, si une variable possède, à un instant donné, une valeur bien définie, celle-ci peut, néanmoins, évoluer au cours de l'exécution du programme.

1.5. Ecriture des noms de variable

Jusqu'ici, nous avons rencontré des variables dont le nom était formé d'une lettre. En fait, vous pouvez également utiliser des noms formés d'une lettre suivie d'un chiffre comme :

```
A5 B4 X9
```

et donc écrire des instructions telles que :

```
LET A5 = 3
LET X9 = A5 - 8
```

Par contre :

```
LET 5X = 8
```

ne serait pas acceptée car 5X n'est pas un nom de variable correct (il ne commence pas par une lettre).

REMARQUE :

De nombreux systèmes autorisent plus de liberté dans le choix des noms de variables. Nous en reparlerons dans les chapitres décrivant le «Basic étendu» car cela peut poser quelques problèmes en induisant des erreurs peu faciles à déceler. Dans l'immédiat, nous vous conseillons de n'utiliser que des noms formés soit d'une lettre, soit d'une lettre suivie d'un chiffre.

2. LES CONSTANTES

2.1. Introduction

Dans l'instruction :

```
10 LET A = 4
```

4 représente une valeur bien définie. On dit qu'il s'agit d'une «constante».

En fait, nous avons déjà rencontré beaucoup de constantes, soit dans des instructions LET, soit dans des instructions PRINT. Ainsi, dans :

```
LET A = C + 5            5 est une constante
LET X = 3.25 * Y         3.25 est une constante
PRINT 3 * B + 1.45       3 et 1.45 sont des constantes
```

2.2. Attention aux mathématiques

Là encore, il faut prendre garde à ne pas confondre cette notion avec celle de constante mathématique, laquelle peut être représentée ou non par une lettre. Ainsi, dans :

$$ax^2 + bx + 5 = 0$$

a, b, 5 et 0 sont des constantes mathématiques.

2.3. Écriture des constantes

Vous avez déjà rencontré des constantes entières telles que 4, ou décimales telles que 3.14. D'une manière générale, vous avez trois façons d'écrire les constantes en Basic :

– sous forme entière : c'est un nombre entier relatif avec ou sans signe, comme :

42 – 35 + 129

– sous forme décimale, avec ou sans signe, le point remplaçant la virgule usuelle. En voici quelques exemples :

32.4 – 410.3 + 3.54

– sous forme exponentielle (ou scientifique). C'est une notation employée sur certaines calculatrices. Elle permet de simplifier l'écriture des grands ou des petits nombres. C'est ainsi que :

2 E 8 s'interprête comme 2.10^8 soit 200000000
0.12 E – 7 s'interprête comme $0.12.10^{-7}$ soit 0,000000012

3. L'INSTRUCTION LET

3.1. Rappels et vocabulaire

L'instruction LET permet donc d'attribuer une valeur à une variable. Là encore, ce terme «attribuer» paraît naturel. Malheureusement, les informaticiens emploient plutôt le mot «affecter». Ils disent que LET est une instruction d'affectation d'une valeur à une variable. Cette valeur peut être une constante comme dans :

```
LET Y = 9
```

ou le résultat d'un calcul comme dans :

```
LET C = X + 3
```

Cette dernière instruction peut s'interprêter ainsi :

1) effectuer le calcul indiqué, ici : X + 3

2) ranger le résultat ainsi obtenu dans la variable C.

3.2. Attention aux mathématiques

Vous voyez que le signe = ne doit surtout pas être interprété comme en mathématiques où il traduit généralement une «relation». Ainsi, par exemple :

$$a = 2\ b$$

signifie que la valeur de a sera toujours le double de celle de b.

Par contre, en Basic :

```
LET A = 2 * B
```

affecte bien à A une valeur double de celle de B, mais rien ne nous dit que cela restera vrai tout au long de l'exécution du programme. Examinez cet exemple :

Instruction	A	B
10 LET B = 3	–	3
20 LET A = 2 * B	6	3
30 LET B = 5	6	5

Après exécution de l'instruction 20, nous avons bien deux fois la valeur de B dans A. Par contre, après exécution de l'instruction 30, ceci n'est plus vrai puisque la valeur de B a changé tandis que celle de A est restée la même.

Voici un autre exemple :

Instruction	C	X
10 LET C = 5	5	–
20 LET X = 3 * C	5	15
30 LET C = 2	2	15

Enfin, considérez les instructions suivantes :

Instruction	A
10 LET A = 5	5
20 LET A = A + 1	6

L'instruction 20 signifie : calculer la valeur de A + 1 (ici A + 1 = 6) et ranger le résultat dans A. Autrement dit, cette instruction a pour effet d'augmenter de 1 la valeur de la variable A.

Ce dernier exemple devrait vous convaincre définitivement de ce que le signe = employé dans l'instruction LET ne traduit pas une relation (mathématiquement, la relation a = a + 1 est impossible), mais bien une affectation instantanée.

3.3. Ne pas confondre la gauche et la droite

Nous avons dit que :

```
LET C = X * Y + 1
```

signifiait : effectuer le calcul mentionné à droite du signe égal et ranger le résultat dans la variable citée à gauche. Vous voyez donc que pour qu'une instruction LET ait une signification, il est nécessaire de trouver le nom d'une variable à gauche du signe égal. Ainsi, ces instructions seraient incorrectes :

```
LET 3 = C    (on aura probablement voulu écrire : LET C = 3)
LET C + 5 = A + 2
```

4. LES EXPRESSIONS ARITHMÉTIQUES

4.1. Définition

A droite du signe égal de l'instruction LET, nous pouvons trouver :

– soit une constante comme dans :

```
LET C = 3
```

– soit l'expression d'un calcul comme dans :

```
LET X = B + 5
```

Nous résumons cela en disant qu'on peut y trouver :

UNE EXPRESSION ARITHMÉTIQUE

Cela correspond à la notion mathématique d'expression algèbrique. Une expression arithmétique exprime un calcul à effectuer. Celui-ci peut porter sur des constantes ou des variables. Les différentes opérations y sont exprimées par des symboles que nous avons déjà rencontré (+, –, *, /, ^).

4.2. Priorités dans le calcul des expressions arithmétiques

Il vous paraît certainement naturel que les instructions :

```
LET A = 3
LET B = 2 * A + 5
PRINT B
```

vous impriment la valeur 11. En effet, dans l'expression arithmétique : 2 * A + 5 Basic a d'abord effectué le produit de 2 par A avant d'ajouter 5 au résultat (si les opérations avaient été réalisées dans l'ordre inverse, nous aurions obtenu 16). On exprime cela en disant que la multiplication a priorité sur l'addition.

De même, dans le calcul de l'expression :

```
A/B – 3
```

Basic effectue d'abord la division de A par B avant de soustraire 3 du résultat. Enfin, dans :

```
A ^2 + C
```

l'élévation à la puissance est effectuée avant l'addition.

D'une manière générale, Basic respecte les priorités suivantes :

1) élévation à la puissance (^)
2) multiplication (*) et division (/)
3) addition (+) et soustraction (–)

Considérez alors l'expression :

```
A/B/C
```

dans laquelle apparaissent deux divisions qui ont donc même niveau de priorité. Vous remarquez certainement que le résultat va dépendre de celle qui sera exécutée en premier. Basic utilise une seconde règle qui dit que, en cas de priorités identiques, les opérations seront exécutées en lisant l'expression de gauche à droite. Ici donc, Basic calcule d'abord A/B. Ce résultat est à son tour divisé par C. Cela correspond finalement à la formule algébrique :

$$\frac{\frac{A}{B}}{C} \quad \text{ou encore} \quad \frac{A}{B \times C}$$

4.3. Exemple : calcul du volume d'un cône

Voici la liste et un exemple d'exécution d'un programme calculant le volume d'un cône, à partir de ses dimensions qui sont fournies au clavier.

```
10 PRINT "ENTRER LE RAYON DE LA BASE"
20 INPUT R
30 PRINT "ENTRER LA HAUTEUR"
40 INPUT H
50 LET V =3.14159*R^2*H/3
60 PRINT "VOLUME=",V
Ok
RUN
ENTRER LE RAYON DE LA BASE
? 2.5
ENTRER LA HAUTEUR
? 7
VOLUME=          45.8148
Ok
```

4.4. Quand les règles de priorité ne suffisent plus. Les parenthèses

Considérons l'expression :

A * X + B/C

D'après ce que nous avons dit, nous voyons que son calcul se déroule ainsi :

1) A est multiplié par X
2) B est divisé par C
3) les deux résultats précédents sont ajoutés

En algèbre, ce calcul s'exprimerait par une expression du type :

$$ax + \frac{b}{c}$$

Supposons maintenant que l'on souhaite écrire en Basic une expression arithmétique correspondant à :

$$\frac{ax + b}{c}$$

Après quelques essais, vous découvrez rapidement que les règles précédemment décrites ne suffisent pas. Il nous faut, en effet, forcer Basic à effectuer certains calculs en premier, contrairement aux règles de priorité. Pour cela, nous utilisons des parenthèses (comme en mathématiques). Toute expression placée entre parenthèses est d'abord évaluée avant de participer à la suite des calculs. Ainsi, pour revenir à notre dernière expression algèbrique, en Basic :

```
(A * X + B)/C
```

répondra à la question posée.

Vous pouvez utiliser plusieurs «niveaux» de parenthèses. Ainsi, cette expression :

```
((A + B/C) * D + E) * F
```

est correcte. Les calculs s'y déroulent ainsi :

1) division de B par C

2) ajout de A

3) multiplication par D

4) ajout de E

5) multiplication par F

Par contre, vous n'avez droit qu'aux deux symboles «(» et «)». Vous ne pouvez pas, comme en mathématiques, utiliser des crochets ou des accolades.

4.5. Attention aux habitudes de l'algèbre

Lorsqu'en algèbre, vous écriviez :

$$3\ x$$

cela signifiait 3 multiplié par x. De même :

$$(x + 2)\ (x - 3)$$

signifiait x + 2 multiplié par x − 3

Autrement dit, vous pouviez, dans certains cas, omettre le signe "multiplié". En Basic, cela est IMPOSSIBLE. Ainsi :

A3

représente la variable nommée A3 et jamais A * 3

Dans l'instruction :

Y = 3 A * X

3A sera considéré comme un nom de variable incorrect.

De même, l'instruction :

LET A = (X + 2) (X – 3)

sera considérée comme érronée car aucune opération n'est mentionnée entre les deux paires de parenthèses.

4.6. Exemple : programme de calcul de l'aire d'un trapèze

```
10 PRINT "ENTREZ LA PREMIERE BASE"
20 INPUT B1
30 PRINT "ENTREZ LA DEUXIEME BASE"
40 INPUT B2
50 PRINT "ENTREZ LA HAUTEUR"
60 INPUT H
70 LET S = (B1+B2)*H/2
80 PRINT "SURFACE=",S
Ok
RUN
ENTREZ LA PREMIERE BASE
? 4
ENTREZ LA DEUXIEME BASE
? 7
ENTREZ LA HAUTEUR
? 3.6
SURFACE=          19.8
Ok
```

4.7. Remarque importante

Nous avons introduit l'expression arithmétique à propos de l'instruction LET. Cependant, vous l'avez constaté dans le chapitre précédent, il nous est arrivé également d'écrire des instructions telles que :

```
PRINT A + B
PRINT 2 * X – 5
```

D'une manière générale, vous pouvez utiliser n'importe quelle instruction arithmétique dans une instruction PRINT.

5. ATTENTION AUX VARIABLES NON DÉFINIES

Considérez ce programme de deux instructions :

```
10 LET B = A + 1
20 PRINT B
```

Lorsque Basic exécute l'instruction 10, la variable A n'a pas encore reçu de valeur. On dit qu'elle n'est pas définie.

La plupart des Basic considèrent qu'une variable non définie contient en fait la valeur 0. Dans notre précédent exemple, l'instruction 10 affecte alors la valeur 1 à B.

Par contre, dans ce programme :

```
10 LET B = 2 * A
20 LET A = 4
30 PRINT A, B
```

quand Basic exécute l'instruction 10, A n'est pas encore définie. B prendra donc la valeur 0, bien que l'on vienne ensuite affecter la valeur 4 à A. Les valeurs écrites par ce programme seront donc 4 et 0.

EXERCICES

1. VRAI ou FAUX ?

a) le contenu d'une variable reste toujours constant pendant l'exécution d'un programme.

b) en informatique, le mot variable désigne, à un instant donné, une valeur quelconque.

c) on peut modifier le contenu d'une variable.

d) on peut ranger en même temps plusieurs valeurs différentes dans une même variable.

e) à chaque fois que l'on imprime le contenu d'une variable, sa valeur est définitivement perdue.

f) dans ce programme :

```
10 LET B = 2
20 LET C = 3
30 LET C = B + 4
```

l'exécution de l'instruction 30 fait disparaître la valeur contenue dans B.

g) dans le programme précédent, l'exécution de l'instruction 30 fait disparaître la valeur contenue dans C.

2. Quels sont les noms de variables corrects :

L1   XA   R7   X.   6   4T   Z   VAL

3. VRAI ou FAUX ? L'instruction :

```
10 LET X = A
```

a) dit que X contiendra toujours la même valeur que A.

b) dit de donner à X la valeur de A.

c) détruit la valeur de A.

4. L'instruction suivante est-elle correcte ?

```
LET N = N + 2
```

5. Que fait l'instruction :

```
LET C = A * X + 3
```

6. Que fait l'instruction :

```
LET A * X + B = 0
```

7. Quelles sont les instructions erronées ?

a) LET C = 5 + X

b) LET 3 = B

c) LET A * X ^2 + B * X + C = 0

8. Quelles seront les valeurs des variables à la fin de chacune des instructions suivantes :

```
10 LET F5 = 4
20 LET G = F5 – 3
30 LET F5 = G
```

9. Même question pour :

```
10 LET X = 4
20 LET Y = 10
30 LET X = Y
40 LET Y = X
```

10. Même question avec :

```
10 LET X = 3
20 LET Y = 10
30 LET W = X
40 LET X = Y
50 LET Y = W
```

Que font les instructions 30 à 50 ?

11. Quels résultats fournira le programme suivant :

```
10 LET A = 2
20 LET B = 5
30 LET A = B + 2 * A
40 LET B = A + B/5
50 PRINT A, B
```

12. L'expression arithmétique est-elle une instruction Basic ?

13. Où pouvez-vous utiliser une expression arithmétique ?

14. VRAI ou FAUX ? Le symbole * peut être omis dans certains cas.

15. Quelles sont les erreurs commises dans les instructions suivantes (supposées indépendantes les unes des autres) :

```
10 LET Z = A * X/3 B
20 LET A5 = 3 (A * X – 5)
30 LET A * B = 8
40 LET (A + 7) * (X – 5)
50 LET Z = AB – 7
60 PRINT Y = 15
70 PRINT Y 3 – 12
80 PRINT (A + B) * 9 * (Y ^2 – 4)
90 PRINT A . X
95 PRINT 3 * A, 5 * C ^2 – 4, (Y + 5) * (Z – 3)
```

16. Qu'est-ce qu'une variable non définie ?

17. Ecrire un programme calculant le volume d'une sphère à partir de son rayon.

VI
LES CHAÎNES DE CARACTÈRES

1. LES CONSTANTES "CHAÎNES DE CARACTÈRES"

En réponse à :

```
PRINT 5
```

vous obtenez :

```
5
```

Si maintenant vous frappez :

```
PRINT "BONJOUR"
```

vous obtenez :

```
BONJOUR
```

Dans le premier cas, vous avez demandé d'écrire le nombre 5. Dans le second cas, vous avez demandé d'écrire les sept lettres :

```
BONJOUR
```

Nous avons déjà dit que 5 était une constante. Pour préciser qu'il s'agit d'un nombre, nous l'appellerons dorénavant une «constante numérique». De la même façon, nous dirons que BONJOUR est une constante «chaîne de caractères» car elle correspond à une suite de caractères bien définis.

Peut-être le mot «caractère» ne vous est-il pas familier. On appelle ainsi n'importe quel symbole qu'il est possible de taper au clavier. On y trouve, bien sûr les 26 lettres de l'alphabet (souvent en majuscules et en minuscules), les dix chiffres décimaux, mais aussi d'autres symboles tels que :

. , ; : + – * = ^ ( ) " etc...

Frappons les instructions (en mode direct) :

```
LET A3 = 10
PRINT A3
```

nous obtenons :

```
10
```

Frappons maintenant :

```
PRINT "A3"
```

nous obtenons :

```
A3
```

En effet, nous avons, cette fois, demandé d'écrire la chaîne de caractères A3 alors qu'auparavant, nous avions demandé d'écrire le contenu de la variable A3.

Vous devez maintenant comprendre la nécessité des guillemets dans les constantes chaînes de caractères. (On les appelle souvent, plus brièvement, des constantes chaînes).

Attention aux chaînes trop longues

Les premiers Basic autorisaient des chaînes possédant au maximum 18 caractères. Actuellement, les limites sont généralement beaucoup plus élevées (souvent de 255 caractères). Toutefois, nous vous conseillons de vérifier ce à quoi vous avez droit sur votre machine.

2. LES VARIABLES "CHAÎNES DE CARACTÈRES"

Lorsque nous écrivons :

```
LET C = 5
```

C désigne une variable destinée à contenir des nombres. Nous dirons que C est une «variable numérique». De la même façon, nous aurons des variables destinées à contenir des chaînes de caractères que nous nommerons «variables chaînes».

Par exemple :

```
LET M$ = "BONJOUR"
PRINT M$
```

nous fournit :

```
BONJOUR
```

M$ est une variable chaîne. L'instruction :

```
LET M$ = "BONJOUR"
```

y range la constante chaîne BONJOUR. On dit également que cette instruction affecte la constante chaîne BONJOUR à la variable M$.

D'une manière générale, le nom d'une variable chaîne est formé d'une lettre suivie éventuellement d'un chiffre et obligatoirement du symbole $. Par exemple :

```
A$    B2$    X$    M9$
```

sont des noms corrects de variables chaînes.

Par contre :

```
AS    B    Z
```

n'en sont pas car ils ne sont pas suivis du symbole $. (Ce sont cependant des noms corrects de variables numériques)

Enfin

```
AB$    5X$
```

sont incorrects

3. L'INSTRUCTION D'AFFECTATION POUR LES VARIABLES CHAÎNES

Nous en avons vu un exemple dans :

LET M$ = "BONJOUR"

Considérez ce programme et son exécution :

```
10 LET N1$ = "DUPONT"
20 LET N2$ = N1$
30 PRINT N1$
40 PRINT N2$
Ok
RUN
DUPONT
DUPONT
Ok
```

L'instruction 10 affecte la constante chaîne DUPONT à la variable chaîne N1$. L'instruction 20 affecte à la variable chaîne N2$, le contenu de la variable chaîne N1$.

Vous voyez donc que l'instruction d'affectation permet d'attribuer à une variable chaîne soit la valeur d'une constante chaîne, soit celle d'une autre variable chaîne.

Voici un autre exemple :

```
10 LET N$ = "DUPONT"
20 LET V$ = "ORLEANS"
30 PRINT N$,"HABITE A",V$
Ok
RUN
DUPONT          HABITE A        ORLEANS
Ok
```

Vous devez certainement trouver que les possibilités sont beaucoup moins variées que dans le cas de l'affectation numérique où vous pouviez affecter à une variable le résultat d'un calcul exprimé par une expression arithmétique. Ceci est dû à ce que les valeurs numériques peuvent intervenir dans plusieurs opérations (+, –, *, /, ^). Ces dernières n'ont pas de sens pour les variables chaînes. Toutefois, nous verrons plus tard qu'il est possible, en basic étendu, d'effectuer certaines opérations entre chaînes. Pour l'instant donc, il nous faut nous contenter de cette affectation un peu pauvre et surtout...

4. NE PAS MÉLANGER LE NUMÉRIQUE ET LES CHAÎNES

Autrement dit, il ne faut pas vouloir affecter des chaînes à des variables numériques ou l'inverse. Par exemple, si vous tapez :

```
LET A = "BONJOUR"
```

vous obtiendrez un message d'erreur du genre :

```
Syntax error
```

De même, l'exécution de ce programme :

```
10 LET V$ = "PARIS"
20 LET A = V$
30 PRINT A
```

conduira à :

```
RUN
Syntax error in 20
```

Un examen de l'instruction 20 vous montre que vous avez voulu y affecter la valeur d'une variable chaîne à une variable numérique.

5. POUR INTRODUIRE DES CHAÎNES - INPUT

5.1. Introduction

Vous avez déjà vu comment introduire des informations numériques lors de l'exécution d'un programme grâce à l'instruction INPUT. C'est ce qui se passe dans :

```
INPUT P1
```

Lorsque Basic l'exécute, il vous demande d'entrer au clavier une valeur qu'il affecte à la variable P1.

Considérez maintenant ce programme :

```
10 PRINT "QUEL EST VOTRE NOM"
20 INPUT N$
30 PRINT "DANS QUELLE VILLE HABITEZ VOUS"
40 INPUT V$
50 PRINT "MONSIEUR",N$,"RESIDE A",V$
Ok
```

Si nous l'exécutons en répondant DUVAL et ORLÉANS aux deux questions posées, nous obtenons :

```
RUN
QUEL EST VOTRE NOM
? DUVAL
DANS QUELLE VILLE HABITEZ VOUS
? ORLEANS
MONSIEUR      DUVAL          RESIDE A      ORLEANS
Ok
```

Vous voyez que l'instruction 20 vous a permis de fournir une chaîne de caractères que Basic a affecté à la variable chaîne N$. De même, l'instruction 40 vous a permis d'affecter une chaîne de caractères à V$.

5.2. Et les guillemets ?

Nous n'avons pas placé de guillemets dans notre réponse d'une chaîne à l'instruction INPUT. Aucune ambiguïté n'existait, en effet, pour Basic. Cependant, vous pouvez également placer votre réponse entre guillemets à condition, toutefois, de ne rien écrire en dehors. Ainsi, nous aurions pu répondre de cette manière au programme précédent :

```
RUN
QUEL EST VOTRE NOM
? "DURAND"
DANS QUELLE VILLE HABITEZ VOUS
? "GRENOBLE"
MONSIEUR      DURAND         RESIDE A      GRENOBLE
Ok
```

Les guillemets n'ont, bien sûr, pas été conservés dans la chaîne de caractères affectée, soit à N$, soit à V$. Par contre si, dans votre

réponse, vous utilisez des guillemets et que vous placez des caractères en dehors, vous obtenez :

```
RUN
QUEL EST VOTRE NOM
? "DURAND" C'EST MON NOM
?Redo from start
?
```

Nous avons déjà rencontré ce message d'erreur par lequel Basic vous demande de fournir à nouveau votre réponse à INPUT.

5.3. En cas de mauvaise réponse

De même, il se peut que, par erreur, vous fournissiez une réponse de type chaîne là où Basic attend une valeur numérique. Là encore, vous obtiendrez ce message d'erreur comme dans cet exemple :

```
10 PRINT "QUEL EST VOTRE AGE"
20 INPUT A
30 LET N = 1982-A
40 PRINT "VOUS ETES NE EN";N
Ok
RUN
QUEL EST VOTRE AGE
? 26
VOUS ETES NE EN 1956
Ok
RUN
QUEL EST VOTRE AGE
? 32 ANS
?Redo from start
?
```

6. RETOUR SUR L'INSTRUCTION PRINT

Lorsque plusieurs variables apparaissent dans l'instruction PRINT, les résultats correspondants se trouvent séparés par plusieurs caractères blancs. Ceci provient de ce que Basic prévoit une «tabulatin» de la ligne en zones de (généralement) 16 caractères. Autrement

dit la première valeur à imprimer l'est à partir de la position 1, la suivante à partir de la position 17 et ainsi de suite.

Vous pouvez éviter cette tabulation automatique et obtenir que tous les résultats soient imprimés les uns à la suite des autres. Pour cela, il vous suffit de remplacer chaque virgule séparant les expressions mentionnées dans l'instruction PRINT par un point-virgule (;).

Voici quelques exemples en mode direct :

```
PRINT "O","U","I"
O               U               I
Ok
PRINT "O";"U";"I"
OUI
Ok
PRINT "CECI","EST","UN","EXEMPLE"
CECI            EST             UN              EXEMPLE
Ok
PRINT "CECI";"EST";"UN";"EXEMPLE"
CECIESTUNEXEMPLE
Ok
PRINT "CECI";" EST";" UN";" EXEMPLE"
CECI EST UN EXEMPLE
Ok
```

Vous remarquez que, dans le dernier cas, il a fallu prévoir les caractères blancs (ou espaces) pour que le texte imprimé apparaisse correctement.

Dans cet exemple :

```
10 LET A = 20
20 LET M$ = "VOUS AVEZ"
30 PRINT M$;A;"ANS"
Ok
RUN
VOUS AVEZ 20 ANS
Ok
```

un blanc apparaît avant et après le nombre 20, bien que vous ne l'ayez pas mentionné. Ceci est dû à ce que Basic prévoit toujours un blanc après le nombre et un emplacement pour le signe (s'il s'agit du signe +, il n'est pas imprimé).

De la même façon, nous aurions :

```
PRINT 15;-30;-50
 15 -30 -50
Ok
```

Enfin, considérez ce programme :

```
10 LET A = 20
20 LET M$ = "VOUS AVEZ"
30 PRINT M$;A;
40 PRINT "ANS"
Ok
RUN
VOUS AVEZ 20 ANS
Ok
```

La deuxième instruction PRINT a écrit sur la même ligne que la précédente. Ceci est dû à ce que cette dernière est terminée par un point-virgule.

7. RETOUR SUR L'INSTRUCTION INPUT

Il est possible de demander dans une même instruction INPUT des valeurs pour plusieurs variables. En voici des exemples :

```
100 INPUT A, N
150 INPUT R$, A$, C$
300 INPUT X, D$
```

Dans ce cas, vous devez fournir plusieurs réponses, séparées par des virgules. Chacune d'entre elles doit, bien sûr, être du type de la variable correspondante. Dans le cas contraire, Basic vous demandera, là encore, de donner à nouveau toutes vos valeurs par un message du type :

```
Redo from start
```

EXERCICES

1. VRAI ou FAUX ?

a) Les variables chaînes de caractères ne peuvent contenir que des lettres.

b) Les variables chaînes contiennent un seul caractère.

2. Quelle différence y-a-t-il entre les deux constantes suivantes :

9 "ERREUR"

3. Même question avec :

25 "25"

3. Relevez les erreurs commises dans les instructions suivantes :

a) LET Z = "CECI EST UN MESSAGE"

b) LET X$ = 35

c) LET A = "3"

d) LET C$ = "2 HEURES"

e) LET H$ = "2 HEURES" – 5

f) LET C$= "5"
LET H$ = "2 HEURES"
LET X$ = H$ – C$

5. Quelle différence présenteront les résultats des deux programmes suivants ?

```
10 LET M$ = "JANVIER"
20 LET J$ = "30"
30 LET D$ = "DATE"
40 PRINT D$, J$, M$
```

```
10 LET M$ = "JANVIER"
20 LET J = 30
30 LET D$ = "DATE"
40 PRINT D$, J, M$
```

6. Y-a-t-il une erreur dans l'un des programmes suivants :

```
10 LET D = 100382
20 PRINT "DATE" ; D
```

```
10 LET D$ = "10-03-82"
20 PRINT "DATE" ; D$
```

7. On considère le programme :

```
10 PRINT "QUELLE ANNEE SOMMES NOUS"
20 INPUT A
```

Que se passera-t-il si nous l'exécutons, en répondant :

a) 1982

b) 80

c) 1815

d) MILLE NEUF CENT QUATRE VINGT DEUX

8. Même question avec le programme :

```
10 PRINT "QUELLE ANNEE SOMMES NOUS"
20 INPUT A$
```

9. Écrire les instructions permettant d'échanger le contenu de deux variables chaînes A$ et B$.

10. Quelle différence y-a-t-il entre les résultats imprimés par ces deux instructions :

```
PRINT "BONJOUR", "MONSIEUR"
PRINT "BONJOUR" ; "MONSIEUR"
```

11. Même question avec :

```
PRINT "BONJOUR" ; ".MONSIEUR"
PRINT "BONJOUR . MONSIEUR"
```

12. Que produira l'exécution de ce programme :

```
10 LET A = – 315
20 LET B = 315
30 LET C = A + B
40 PRINT A ; "+" ; B ; "=" ; C
```

13. Que se passera-t-il, si nous exécutons ce programme :

```
10 PRINT "ENTRER LE RAYON"
20 INPUT R
30 LET S = 3.14159 * R ^2
40 PRINT "SURFACE =" ; S
```

suivant que l'on répond :

a) 10 CM

b) 1 M

c) 10

VII

POUR FAIRE DES CHOIX L'INSTRUCTION IF

Dans tous les programmes que nous avons rencontrés jusqu'ici, les instructions s'exécutaient dans l'ordre croissant des numéros de ligne. Dans ce chapitre, nous allons apprendre à effectuer des choix à l'intérieur d'un programme.

1. EXEMPLE D'INTRODUCTION

Examinez ce programme :

```
100 INPUT N
110 IF N>10 THEN 140
120 PRINT N;"N'EST PAS SUPERIEUR A 10"
130 GOTO 150
140 PRINT N;"EST SUPERIEUR A 10"
150 REM
Ok
```

Exécutons-le plusieurs fois avec différentes réponses :

```
RUN
? 15
 15 EST SUPERIEUR A 10
Ok
RUN
? 5
 5 N'EST PAS SUPERIEUR A 10
Ok
RUN
? 10
 10 N'EST PAS SUPERIEUR A 10
Ok
```

Comment s'est-il comporté ? L'instruction 100 a demandé d'entrer au clavier une valeur numérique et de l'affecter à la variable N. La première fois, nous avons répondu 15. L'instruction 110 est nouvelle pour vous. Elle signifie si (en anglais IF) la valeur de N est supérieure à 10, aller exécuter l'instruction 140. Comme ici N contient 15, on a bien N > 10 et Basic va donc exécuter l'instruction 140. Celle-ci nous écrit la valeur de N suivie du texte «EST SUPERIEUR A 10». Basic passe alors à l'instruction suivante, de numéro 150. C'est une remarque et par suite, elle est sans effet. On se trouve alors avoir atteint la fin du programme (notez que nous aurions pu tout aussi bien placer, en 150, une instruction END).

Dans la deuxième exécution, nous avons fourni pour N la valeur 5. L'instruction 110 a de nouveau comparé cette valeur à 10. Cette fois, l'affirmation N > 10 est fausse. Par suite, Basic ne va pas à l'instruction 140 mais il passe simplement à l'instruction suivante (on dit souvent qu'il va «en séquence»). L'instruction 120 nous écrit la valeur de N suivie du texte «N'EST PAS SUPERIEUR A 10». L'instruction 130 est également nouvelle pour vous. Elle signifie : aller à (en anglais GO TO) l'instruction 150. On se trouve alors en fin de programme.

La troisième exécution a le même effet que la seconde puisque l'affirmation 10 > 10 est fausse.

En définitive, le programme ci-dessus peut se comporter d'une manière ou d'une autre suivant la valeur que vous lui fournirez pour N. Cette possibilité d'effectuer des choix (on dit aussi souvent prendre des décisions) basés sur des comparaisons de variables est l'un des éléments qui donnent une certaine puissance aux programmes.

2. BIEN PRÉSENTER SES PROGRAMMES POUR Y VOIR PLUS CLAIR

Tant que vos programmes ne comportent que des instructions s'exécutant dans l'ordre où elles apparaissent dans la liste, vous n'avez généralement pas trop de difficultés pour voir ce qu'ils font. Par contre, dès que ceci n'est plus vrai, il peut être judicieux d'expliquer les choix faits à l'aide de remarques. Par exemple, dans le programme précédent (certes encore très simple), nous pourrions ajouter quelques instructions REM, de la façon suivante :

```
100 REM PROGRAMME COMPARANT UN NOMBRE A 10
110 REM
120 REM INTRODUCTION D'UN NOMBRE AU CLAVIER
130 INPUT N
140 REM CE NOMBRE EST-IL SUPERIEUR A 10 ?
150 IF N > 10 THEN 190
160 REM NON
170 PRINT N;"N'EST PAS SUPERIEUR A 10"
180 GOTO 210
190 REM OUI
200 PRINT N;"EST SUPERIEUR A 10"
210 REM
Ok
```

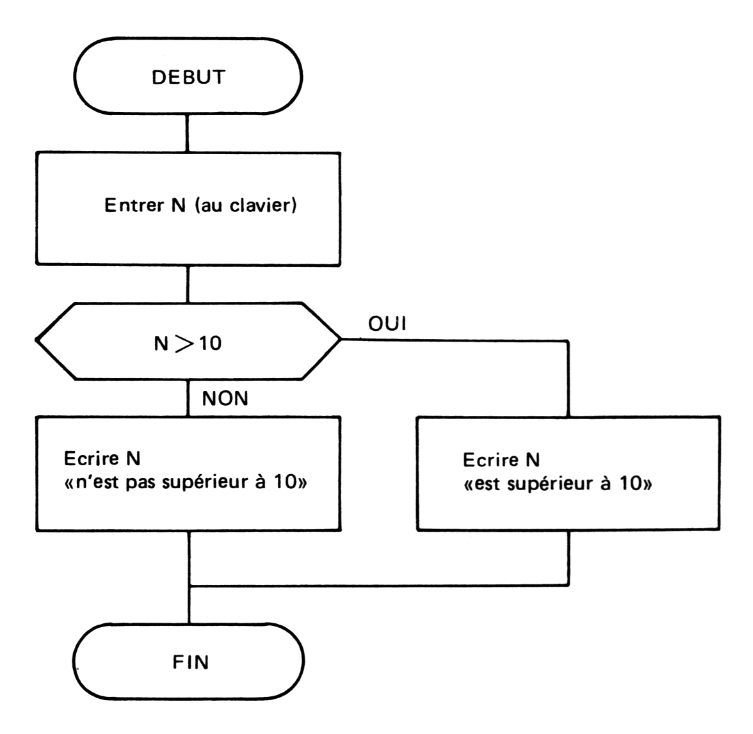
Il se peut que vous trouviez cette présentation un peu lourde. Ceci est dû, en partie, à l'absence de distinction entre les remarques et les autres instructions. Vous pouvez améliorer la situation en utilisant la possibilité offerte par Basic d'ajouter des caractères blancs dans vos instructions. Le programme précédent pourrait s'écrire :

```
100 REM .............PROGRAMME COMPARANT UN NOMBRE A 10
110 REM
120 REM .............INTRODUCTION D'UN NOMBRE AU CLAVIER
130    INPUT N
140    REM...........CE NOMBRE EST'IL SUPERIEUR A 10 ?
150    IF N > 10 THEN 190
160    REM...........NON
170       PRINT N;"N'EST PAS SUPERIEUR A 10"
180       GOTO 210
190    REM...........OUI
200       PRINT N;"EST SUPERIEUR A 10"
210 REM
Ok
```

Nous avons décalé de trois caractères le début des instructions autres que REM. En outre, toutes les instructions correspondant au choix fait en 150 ont été décalées de trois caractères supplémentaires. Cette façon de faire permet de voir d'un seul coup d'œil toutes les instructions concernées par le choix.

3. UN DESSIN VAUT-IL MIEUX QU'UN LONG DISCOURS ?

Pour mieux comprendre ce que fait un programme, vous pouvez réaliser un diagramme en montrant les enchaînements logiques : c'est ce que l'on appelle un organigramme. Il existe d'assez nombreux symboles pouvant intervenir dans de tels schémas. Nous nous limiterons aux principaux. Un organigramme du programme précédent pourrait être :



A l'exception des symboles DEBUT et FIN, nous avons utilisé :
– le rectangle qui représente une ou plusieurs «activités» parmi :

* entrée d'information (INPUT)
* écriture d'information (PRINT)
* calculs (LET)

– l'hexagone allongé qui précise que l'on fait un choix. A l'intérieur se trouve une condition. Les mots OUI et NON montrent quel chemin emprunter suivant que cette condition est vraie ou fausse.

Malgré son aspect séduisant, l'organigramme présente un énorme défaut : c'est celui d'être extérieur au programme. Autrement dit, il exprime d'une nouvelle manière ce que doit faire un programme. Dans ces conditions, il arrive que les deux ne soient pas en accord. Cela peut se produire lorsque l'on modifie une partie d'un programme en omettant de corriger l'organigramme correspondant.

Nous ne vous conseillerons donc pas de faire systématiquement des organigrammes. Nous pensons que des remarques bien placées, une bonne «mise en page» et de bonnes habitudes de programmation sont de meilleurs atouts pour parvenir à une certaine maîtrise de vos programmes. Cependant, il vous faut connaître les organigrammes pour au moins trois raisons :

– au début, ils peuvent vous paraître pratiques.

– même si vous ne vous en servez pas, vous risquez d'en rencontrer dans certains manuels.

– nous les utiliserons (certes modérement) pour «visualiser» certaines notions.

4. UN PETIT PROBLÈME D'IMPÔTS

Voici un programme qui demande d'entrer au clavier :

– le salaire annuel S

– les dépenses D faites pour l'amélioration de l'habitat

Le revenu imposable R est obtenu en réduisant de 10 % puis de 20 % le salaire S (ce qui revient, en fait, à multiplier S par 0,9 × 0,8 = 0,72), puis en déduisant les dépenses D plafonnées à

7000 francs. L'impôt I sera alors calculé (de façon très simpliste), comme suit :

– si R <= 30000   I = 0

– si R >   30000   I = S x 0,15

```
100 REM ********** CALCUL SIMPLIFIE D'IMPOT **************
110 REM
120 REM                S = SALAIRE ANNUEL
130 REM                D = DEPENSES FAITES POUR L'HABITAT
140 REM                R = REVENU IMPOSABLE
150 REM                I = IMPOT
160 REM
170 REM ----------------ENTREE DES VALEURS DE S ET D AU CLAVIER
180 REM
190    PRINT "QUEL EST VOTRE SALAIRE ANNUEL NET ?"
200    INPUT S
210    PRINT "QUELLES SONT VOS DEPENSES POUR L'HABITAT ?"
220    INPUT D
230 REM
240 REM -----------------CALCUL DU REVENU IMPOSABLE
250    LET R = S*.72
260    IF D < 7000 THEN 280
270       LET D = 7000
280    REM
290    LET R = R-D
300 REM
310 REM -----------------CALCUL DE L'IMPOT
320     IF R < 30000 THEN 360
330     REM ........REVENU IMPOSABLE SUPERIEUR A 30000
340        LET I = .15*(R-30000)
350        GOTO 380
360     REM ........REVENU IMPOSABLE INFERIEUR OU EGAL A 30000
370        LET I = 0
380     REM
390 REM -----------------ECRITURE DE L'IMPOT
400     PRINT "IMPOT=";I
Ok
```

```
RUN
QUEL EST VOTRE SALAIRE ANNUEL NET ?
? 40000
QUELLES SONT VOS DEPENSES POUR L'HABITAT ?
? 0
IMPOT= 0
Ok
RUN
QUEL EST VOTRE SALAIRE ANNUEL NET ?
? 100000
QUELLES SONT VOS DEPENSES POUR L'HABITAT ?
? 5000
IMPOT= 5550
Ok
RUN
QUEL EST VOTRE SALAIRE ANNUEL NET ?
? 100000
QUELLES SONT VOS DEPENSES POUR L'HABITAT ?
? 15000
IMPOT= 5250
Ok
```

5. POUR COMPARER DES CHAINES

Les choix que nous avons programmés jusqu'ici étaient basés sur la comparaison de valeurs numériques. Il est également possible de comparer des chaînes de caractères. Examinez ce programme :

```
10 PRINT "ENTREZ 2 NOMS (SEPARES PAR UNE VIRGULE)"
20 INPUT N1$,N2$
30 IF N1$ > N2$ THEN 60
40    PRINT N1$;" EST AVANT ";N2$
50    GOTO 80
60 REM
70    PRINT N1$;" EST APRES ";N2$
80 REM
Ok
```

Exécutons-le avec diverses réponses :

```
RUN
ENTREZ 2 NOMS (SEPARES PAR UNE VIRGULE)
? TABLE,CHAISE
TABLE EST APRES CHAISE
Ok
RUN
ENTREZ 2 NOMS (SEPARES PAR UNE VIRGULE)
? TABLE,TABLEAU
TABLE EST AVANT TABLEAU
Ok
RUN
ENTREZ 2 NOMS (SEPARES PAR UNE VIRGULE)
? PARIS1,PARIS2
PARIS1 EST AVANT PARIS2
Ok
```

Comment s'est-il comporté ? Les instructions 10 et 20 ont permis d'entrer au clavier deux chaînes de caractères dans les variables N1$ et N2$. L'instruction 30 demande d'aller en 50 si le contenu de N1$ est supérieur à celui de N2$. Cette comparaison peut surprendre lorsqu'elle porte sur des chaînes de caractères. Si l'on examine les résultats obtenus, on constate que Basic considère que :

"TABLE" > "CHAISE" est vrai
"TABLE" > "TABLEAU" est faux

Ces chaînes ont, en fait, été comparées suivant l'ordre alphabétique. Ainsi N1$ > N2$ signifie que le contenu de N1$ arrive après celui de N2$, par ordre alphabétique. Ceci explique que la première affirmation soit vraie alors que la seconde est fausse.

On constate ensuite que :

"PARIS 1" > "PARIS 2" est faux

Lorsque dans la vie courante, nous parlons d'ordre alphabétique, nous sous-entendons généralement qu'il s'agit des 26 lettres de l'alphabet. En Basic, tous les caractères peuvent êtres classés. C'est ainsi que les chiffres 0 à 9 sont rangés dans cet ordre, et situés avant les lettres. De même, le caractère blanc est placé avant ces chiffres. Quant aux autres caractères, nous en reparlerons plus tard...

6.1. Ranger deux nombres par ordre croissant

Supposez que l'on souhaite écrire un programme qui range deux nombres (entrés au clavier) par ordre croissant. On peut, par exemple, après avoir demandé d'entrer deux valeurs dans deux variables N1 et N2, comparer ces deux dernières. Si N2 > N1, on échange leurs valeurs (c'est-à-dire que la valeur de N1 se retrouve dans N2 et réciproquement). Dans le cas contraire, on ne fait rien. Vous vous souvenez peut-être que, pour échanger le contenu de deux variables, il ne faut surtout pas écrire :

```
LET N2 = N1
LET N1 = N2
```

ce qui affecterait aux deux variables N1 et N2 la valeur contenue au départ dans N1, l'ancienne valeur de N2 étant détruite. Il est nécessaire, pour obtenir le résultat souhaité, d'utiliser une troisième variable, par exemple X.

Le programme pourrait être :

```
100 REM   PROGRAMME RANGEANT DEUX NOMBRES
110 REM          PAR ORDRE CROISSANT
120 REM
130    PRINT "DONNEZ 2 NOMBRES"
140    INPUT N1,N2
150 REM
160 REM ------ COMPARAISON ET INVERSION EVENTUELLE ----
170    IF N1 < N2 THEN 220
180    REM ..... N1 > N2 ... INVERSION
190       LET X = N1
200       LET N1 = N2
210       LET N2 = X
220    REM
230 REM -------- ECRITURE DES RESULTATS --------
240    PRINT "PAR ORDRE CROISSANT";N1;N2
Ok
```

Vérifions sur quelques cas qu'il fonctionne correctement :

```
RUN
DONNEZ 2 NOMBRES
? 25,12
PAR ORDRE CROISSANT 12  25
Ok
RUN
DONNEZ 2 NOMBRES
? 15,22
PAR ORDRE CROISSANT 15  22
Ok
RUN
DONNEZ 2 NOMBRES
? 23.4,128.45
PAR ORDRE CROISSANT 23.4  128.45
Ok
```

6.2. Ranger deux noms par ordre alphabétique

Le problème est très voisin du précédent. Il suffit, en effet de remplacer les trois variables numériques N1, N2 et X par trois variables chaînes, par exemple N1$, N2$ et X$. Si on le désire, on peut aussi adapter le message pour qu'il demande deux noms au lieu de deux nombres. Le programme pourrait être :

```
100 REM   PROGRAMME RANGEANT DEUX NOMS
110 REM         PAR ORDRE ALPHABETIQUE
120 REM
130    PRINT "DONNEZ DEUX NOMS"
140    INPUT N1$,N2$
150 REM
160 REM    ------ COMPARAISON ET INVERSION EVENTUELLE -----
170    IF N1$ < N2$ THEN 220
180    REM ..... N1$ > N2$ ... INVERSION
190       LET X$  = N1$
200       LET N1$ = N2$
210       LET N2$ = X$
220    REM
230 REM    ----- ECRITURE DES RESULTATS --------
240    PRINT "PAR ORDRE ALPHABETIQUE : ";N1$;" ";N2$
Ok
```

Vérifions le également sur quelques cas :

```
RUN
DONNEZ DEUX NOMS
? TABLE,CHAISE
PAR ORDRE ALPHABETIQUE : CHAISE TABLE
Ok
RUN
DONNEZ DEUX NOMS
? TABLE,TABLEAU
PAR ORDRE ALPHABETIQUE : TABLE TABLEAU
Ok
RUN
DONNEZ DEUX NOMS
? PARIS2,PARIS1
PAR ORDRE ALPHABETIQUE : PARIS1 PARIS2
Ok
```

7. LES COMPARAISONS POSSIBLES

Nous avons déjà vu que l'on pouvait comparer des variables numériques ou caractères.

7.1. Comparaisons numériques

D'une manière générale, vous pouvez comparer deux expressions arithmétiques quelconques, à l'aide des comparateurs :

<	inférieur à
=	égal à
>	supérieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
><	différent de (inférieur ou supérieur)

En voici quelques exemples :

```
 50 IF A + B < C/4 THEN 80
100 IF I + 3 = L THEN 40
200 IF (X + 3)/5 <= (Y - 7) ^2  THEN 250
```

7.2. Comparaisons entre chaînes

Vous pouvez comparer entre elles des variables ou des constantes chaînes de caractères, à l'aide des mêmes comparateurs que ceux décrits ci-dessus. Leur signification devient alors :

- < placé avant dans l'ordre alphabétique
- = égal (deux chaînes sont égales si elles sont formées des mêmes caractères)
- > placé après dans l'ordre alphabétique
- <= placé avant ou égal
- >= placé après ou égal
- <> différent (placé avant ou après)

En voici des exemples :

```
 80 IF X$ = "OUI" THEN 120
200 IF Y$ <> Z6$ THEN 400
```

REMARQUES :

Il est, bien sûr, impossible de comparer une expression numérique avec une chaîne de caractères. Si dans un programme, vous écrivez de telles comparaisons, comme dans :

```
100 IF A + B < C$ THEN 180
```

Basic vous le signalera (lors de l'exécution) par un message du type :

```
Syntax error in 100
```

EXERCICES

1. Que signifie l'instruction :

IF A < 20 THEN 50

2. VRAI ou FAUX ?

a) 100 IF K > 20 THEN 30

Signifie : si la valeur de K est supérieure à 20, affecter à K la valeur 30

b) Un organigramme permet de placer des commentaires dans un programme.

3. Que fait le programme suivant :

```
100 INPUT N
110 IF N = 10 THEN 130
120 LET N = 10
130 PRINT N
```

4. Que peut-on comparer dans une instruction IF ?

5. Trouvez les erreurs commises dans les instructions :

```
10 IF A = C THEN 30
20 IF C$ = Z THEN 100
30 IF X1 = 3 THEN Y
40 IF (X − 8 = Y) THEN 150
50 IF X7 = Z4 = 5 THEN 100
```

6. VRAI ou FAUX ? En Basic, suivant l'ordre «alphabétique», les chiffres arrivent après les lettres.

7. Que fait ce programme :

```
10 INPUT N1$
20 INPUT N2$
30 IF N1$ < N2$ THEN 60
40   PRINT N1$ ; "EST APRES" ; N2$
50 REM
60   PRINT N1$ ; "EST AVANT" ; N2$
```

lorsque l'on lui fournit les réponses suivantes pour N1$ et N2$;

réponse pour N1$	réponse pour N2$
PARIS	TOULOUSE
TOUS	TOUTES
TOUT	TOUTES
" ZOE"	ANNIE
A3	A5

A25	A9
6	B
67	95
48	145

8. Écrire un programme qui demande d'entrer deux valeurs et qui en écrit la plus grande

9. Écrire un programme qui demande d'entrer un nom puis qui pose la question :

ETES-VOUS UN HOMME

et qui, en fonction de la réponse (supposée être toujours OUI ou NON), écrit :

BONJOUR MONSIEUR XXXXXXX

ou

BONJOUR MADAME XXXXXXX

(XXXXXXX représentant le nom fournit).

VIII

POUR RÉPÉTER DES INSTRUCTIONS FOR ET NEXT

Jusqu'ici, dans tous les programmes que nous avons rencontrés, chaque instruction était exécutée au maximum une fois pendant le déroulement du programme. Peut-être d'ailleurs avez-vous eu l'impression, à juste titre, que le temps passé à écrire un programme était long par rapport à ce que vous apportait son exécution. En fait, vous allez apprendre maintenant comment exécuter plusieurs fois un certain nombre d'instructions, sans avoir à les réécrire autant de fois que nécessaire.

1. NOTION DE BOUCLE

1.1. Quand on se répète

Prenons un exemple très simple : on souhaite écrire 10 fois de suite la constante caractère :

```
************
```

afin de dessiner un «rectangle» rempli d'étoiles («*»)

Il serait possible d'écrire 10 fois une instruction comme :

```
100 PRINT "************"
```

(en utilisant, bien sûr, 10 numéros de ligne différents)

On peut cependant essayer de faire mieux en songeant à faire répéter plusieurs fois cette instruction par l'utilisation d'un GO TO. On a alors envie d'écrire :

```
100 PRINT "************"
110 GO TO 100
```

Mais alors, on voit que l'exécution d'un tel programme conduit à répéter indéfiniment l'instruction PRINT. On dit souvent que l'on a une «boucle infinie».

On pense alors à compter le nombre de fois que l'on exécute cette instruction (on dit encore compter le nombre de tours de boucles), en s'arrêtant lorsque le nombre voulu (ici 10) est atteint.

1.2. Apprenons à compter

Voyons tout d'abord le comptage. On peut le réaliser avec une variable dont la valeur augmente de 1 à chaque «tour de boucle». Par exemple :

```
100 PRINT "************"
110 LET N = N + 1
120 GO TO 100
```

On s'aperçoit alors qu'il faut affecter une valeur à la variable N avant de commencer à exécuter les instructions de la boucle. D'où :

```
 90 LET N = 1
100 PRINT "************"
110 LET N = N + 1
120 GO TO 100
```

1.3. Il faut savoir s'arrêter

Pour n'exécuter que dix fois cette boucle, il ne suffit pas d'en compter les tours, il faut placer entre les instructions 90 et 100 un choix entre :

– poursuivre la boucle si N n'a pas encore dépassé la valeur 10

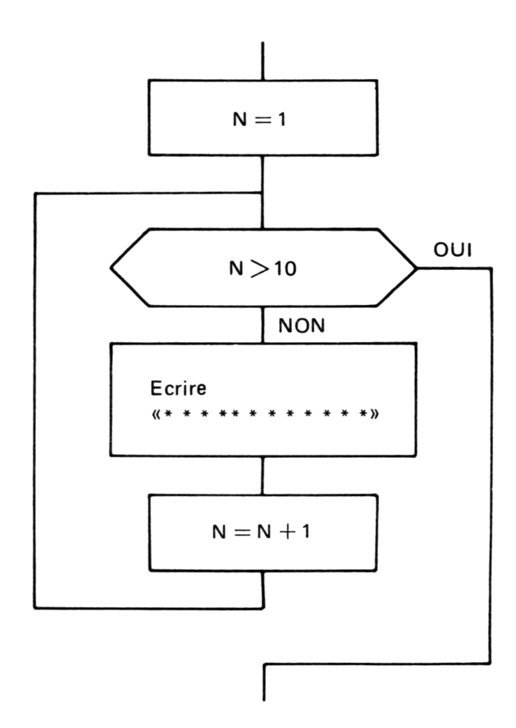
– passer à la suite du programme dans le cas contraire.

Ce choix peut se faire à l'aide d'une instruction IF. Finalement, nous écrivons :

```
 90 LET N = 1
 95 IF N > 10 THEN 130
100 PRINT "************"
110 LET N = N + 1
120 GO TO 95
130 REM
```

1.4. Faisons un dessin

Il est intéressant de faire l'organigramme correspondant aux instructions ci-dessus.



1.5. Ce qu'il faut retenir de cet exemple

Vous voyez que quatre notions nouvelles interviennent dans la réalisation d'une boucle :

– l'utilisation d'un compteur, c'est-à-dire d'une variable comptant le nombre de tours de boucle.

– l'initialisation du compteur, c'est-à-dire l'affectation d'une valeur initiale à cette variable (instruction LET N = 1 de notre exemple)

– l'incrément du compteur, c'est-à-dire l'augmentation de sa valeur à chaque tour de boucle (instruction LET N = N + 1).

– un test d'arrêt : instruction :

```
IF N > 10 THEN 130
```

1.6. Autre écriture d'une boucle. L'instruction FOR

L'exemple précédent peut, en fait, s'écrire beaucoup plus simplement en Basic :

```
 90 FOR N = 1 TO 10
100 PRINT "************"
110 NEXT N
```

L'instruction 90 signifie : tant que N n'a pas dépassé la limite 10, répéter les instructions suivantes (jusqu'à rencontre d'une instruction NEXT N) en augmentant à chaque fois de un la valeur de N.

L'instruction NEXT N ne vous paraît peut-être pas indispensable. Ceci est dû à ce que notre exemple correspond à une boucle ne comportant qu'une seule instruction utile qui est :

```
PRINT "************"
```

En fait, une boucle peut comporter plusieurs instructions utiles. Dans ce cas, le NEXT permet de préciser jusqu'où elle s'étend. Par exemple, dans ce programme :

```
100 FOR X = 1 TO 5
110 PRINT "************"
120 PRINT "----------"
130 NEXT X
```

l'instruction 100 précise qu'on souhaite effectuer une boucle dont le compteur X varie de 1 à 5. Lors de son exécution, Basic attribue à X la valeur 1. Comme cette valeur est inférieure à la limite (5), il exécute les instructions suivantes, soit 110 est 120 qui nous écrivent :

```
************
---------
```

L'instruction 130 incrémente de 1 la variable X qui prend alors la valeur 2. Comme cette valeur est toujours inférieure à 5, Basic exécute à nouveau les instructions 110 à 120. Cela a lieu de la même façon cinq fois de suite. A la fin du cinquième tour de boucle, X vaut alors 5. L'instruction NEXT X lui attribue la valeur 6. Comme, cette fois, X est supérieur à 5, la boucle est terminée et Basic poursuit l'exécution à l'instruction suivant le NEXT X (non représentée ici).

REMARQUES :

1. Lorsque la boucle se termine et que l'on passe à l'instruction suivant le NEXT, le compteur X vaut 6 et non 5, comme on pourrait s'y attendre.

2. Dans cet exemple, il n'était pas indispensable que X varie de 1 à 5. Nous aurions obtenu le même résultat en remplaçant l'instruction 100 par :

```
100 FOR X = 2 TO 6
```

ou encore, par :

```
100 FOR X = 11 TO 15
```

2. CALCUL D'UNE SOMME DE NOMBRES

2.1. Calcul de la somme de 20 nombres

On souhaite écrire un programme calculant la somme de 20 nombres entrés au clavier. On pourrait, pour cela, utiliser 20 variables différentes nommées, par exemple :

```
A1, A2, A3, ..........., A20
```

et écrire une instruction du type :

```
LET S = A1 + A2 + A3 + ...........+ A20
```

qui affecterait bien à la variable S la somme cherchée.

Cependant, cette façon de procéder serait peu élégante et de toute manière inutilisable pour 100 ou 1 000 valeurs et encore moins pour un nombre quelconque de valeurs, non connu à l'avance.

L'idée consiste alors à n'utiliser qu'une variable, par exemple A, destinée à contenir successivement les diverses valeurs entrées au clavier. A chaque fois qu'une nouvelle valeur est entrée, on l'ajoute à une variable S. Ceci doit permettre d'obtenir la somme recherchée, à condition d'avoir préalablement donné la valeur 0 à S. Le programme pourrait être :

```
100 LET S = 0
110 FOR I = 1 TO 20
120     INPUT A
130     LET S = S + A
140 NEXT I
150 PRINT S
Ok
```

Voyons comment il se déroule, en écrivant les instructions dans l'ordre où Basic les exécute avec les valeurs prises par chaque variable (à la fin de l'exécution de chaque instruction) : Nous supposons que les trois premières valeurs entrées au clavier ont été 5, 11 et 2.

Instruction	I	A	S
100 LET S = 0	–	–	0
110 FOR I = 1 TO 20	1	–	0
120 INPUT A	1	5	0
130 LET S = S + A	1	5	5
140 NEXT I	2	5	5
120 INPUT A	2	11	5
130 LET S = S + A	2	11	16
140 NEXT I	3	11	16
120 INPUT A	3	2	16
130 LET S = S + A	3	2	18
etc			

Vous voyez qu'ainsi, les valeurs entrées au clavier sont progressivement ajoutées à S.

2.2. Quand on ne sait pas exactement ce que l'on veut

Le programme précédent sert à additionner 20 valeurs. Si on souhaite pouvoir en additionner un nombre quelconque, on peut procéder ainsi :

```
100 PRINT "COMBIEN DE NOMBRES VOULEZ VOUS ADDITIONNER"
110 INPUT N
120 LET S = 0
130 FOR I = 1 TO N
140    PRINT "ENTRER UN NOMBRE"
150    INPUT A
160    LET S = S + A
170 NEXT I
180 PRINT "SOMME = ";S
Ok
```

L'instruction 110 permet d'entrer une valeur pour N. L'instruction 130 permet de répéter les instructions 140 à 160 un nombre de fois égal à cette valeur. Exécutons ce programme avec diverses valeurs de N.

```
RUN
COMBIEN DE NOMBRES VOULEZ VOUS ADDITIONNER
? 3
ENTRER UN NOMBRE
? 10
ENTRER UN NOMBRE
? 12
ENTRER UN NOMBRE
? 8
SOMME =  30
Ok
RUN
COMBIEN DE NOMBRES VOULEZ VOUS ADDITIONNER
? 1
ENTRER UN NOMBRE
? 21
SOMME =  21
Ok
```

Mais direz-vous, que se passe-t-il si (peut-être par erreur), l'on fournit pour N une valeur négative ou nulle. Essayons le :

```
RUN
COMBIEN DE NOMBRES VOULEZ VOUS ADDITIONNER
? 0
SOMME =  0
Ok
```

Vous constatez que les instructions de la boucle ne sont pas exécutées. Ce résultat était prévisible. En effet, nous avons vu que la comparaison entre le compteur et la limite avait lieu en début de boucle. Dans ces conditions, si, dès le début, le compteur a une valeur inférieure à la limite, aucun tour de boucle n'a effectivement lieu.

2.3. Faisons la moyenne

Supposons que dans notre précédent exemple, les valeurs entrées représentent des notes dont on veut obtenir la moyenne. On voit qu'il suffit pour cela d'en diviser la somme par le nombre de notes, d'où le nouveau programme :

```
100 PRINT "COMBIEN DE NOTES"
110 INPUT N
120 LET S = 0
130 FOR I = 1 TO N
135    PRINT "ENTRER UNE NOTE"
140    INPUT A
150    LET S = S + A
160 NEXT I
170 M = S/N
180 PRINT "MOYENNE =";M
Ok
```

REMARQUE :

Peut-être trouvez-vous regrettable de devoir fournir, au départ, le nombre total de notes. Vous pourriez souhaiter entrer les notes, sans les compter, en précisant seulement quand vous avez terminé. Ceci est tout-à-fait réalisable et nous en reparlerons à propos des boucles conditionnelles. Dans le présent chapitre, nous nous limitons à l'instruction FOR qui ne fait rien d'autre que de compter des tours de boucle.

3. UTILISER LE COMPTEUR

Dans les exemples précédents, la variable «compteur» servait uniquement à compter le nombre de répétitions. Sa valeur n'était pas utilisée en tant que telle dans les instructions situées à l'intérieur de la boucle. Par contre, si nous exécutons ce programme :

```
100 FOR I = 1 TO 5
110     PRINT I
120 NEXT I
Ok
```

nous obtenons :

```
RUN
 1
 2
 3
 4
 5
Ok
```

Vous voyez que nous avons pu, à l'intérieur de la boucle, utiliser (ici, de façon très simple) la valeur du compteur I.

De même, l'exécution de :

```
100 REM    CUBES DES 5 PREMIERS NOMBRES ENTIERS
110     FOR I = 1 TO 5
120        LET I3 = I^3
130        PRINT I,I3
140     NEXT I
Ok
```

fournit :

```
RUN
 1            1
 2            8
 3            27
 4            64
 5            125
Ok
```

4. QUAND ON VEUT UN PAS DIFFÉRENT DE 1

Nous avons appris à réaliser, avec l'instruction FOR, des boucles où la valeur du compteur augmente d'une unité à chaque tour (on dit aussi qu'elle varie par pas de 1). Mais, en fait, on rencontre souvent des problèmes où il serait pratique d'incrémenter le compteur d'une quantité quelconque. Ceci est permis par Basic et nous allons en voir quelques exemples.

4.1. Calcul de la somme des premiers nombres impairs

On veut calculer la somme de tous les nombres impairs allant de 1 à 39. On peut penser à utiliser une boucle où le compteur nommé, par exemple, N prend successivement les valeurs :

1, 3, 5, 7, ..., 37, 39

Autrement dit, la valeur de N doit augmenter de 2 à chaque tour de boucle. Quant au calcul de la somme, il se fait, comme précédemment, en initialisant une variable S à 0 et en lui ajoutant successivement les diverses valeurs prises par le compteur.

La seule chose nouvelle est donc l'écriture d'une instruction FOR permettant d'incrémenter de 2 le compteur à chaque tour. Voici le programme correspondant :

```
100 REM    SOMME DES 20 PREMIERS NOMBRES IMPAIRS
110    LET S = 0
120    FOR N = 1 TO 39 STEP 2
130       LET S = S + N
140    NEXT N
150    PRINT "SOMME =";S
Ok
RUN
SOMME = 400
Ok
```

Vous voyez la nouvelle façon d'écrire l'instruction FOR. STEP 2 signifie que l'on utilise un pas de 2 pour le compteur. Dans les exemples précédents, aucun pas n'était spécifié et Basic considérait alors que sa valeur était de 1.

4.2. Calcul des valeurs d'une fonction pour plusieurs valeurs de la variable

On souhaite calculer la quantité :

$$y = x^3 + 4\,x^2 - 5$$

pour les valeurs suivantes de x :

0 0,2 0,4 0,6 .... 1,8 2

On peut utiliser une boucle où le compteur X représente la valeur de x. La valeur de départ est 0, celle de fin est 2 et le pas est de 0,2. D'où le programme :

```
100 REM   CALCUL DE X^3 + 4X^2 - 5
110    FOR X = 0 TO 2 STEP .2
120       LET Y = X^3 + 4*X^2 - 5
130       PRINT X,Y
140    NEXT X
Ok
```

Il s'agit de la même écriture de l'instruction FOR que dans l'exemple précédent ; mais, cette fois, le pas n'en est plus entier. Exécutons ce programme :

```
RUN
 0             -5
 .2            -4.832
 .4            -4.296
 .6            -3.344
 .8            -1.928
 1              0
 1.2            2.488
 1.4            5.584
 1.6            9.336
 1.8            13.792
Ok
```

Nous découvrons que, bien qu'ayant demandé que le compteur varie de 0 à 2, la valeur 2 n'a jamais été considérée. Pour comprendre ce phénomène, il faut savoir que les nombres sont représentés en mémoire sous forme codée (en binaire). La plupart du temps, cela conduit à une valeur approchée. Nous reviendrons sur ce point dans le chapitre 18. Pour l'instant, vous pouvez imaginer que lors du codage de 0,2 en mémoire, il se passe quelque chose d'analogue à ce que vous rencontrez lorsque vous voulez écrire 2/3 sous forme

d'un décimal à 6 chiffres. Vous ne pouvez, en effet, que choisir entre 0,666666 ou 0,666667 qui n'en sont que des valeurs approchées.

Revenons à notre exemple. Supposons que la valeur codée de 0,2 soit légèrement trop grande, par exemple 0,2000001. Alors, après avoir été incrémentée dix fois, la valeur de X sera environ 2,000001. Comme cette valeur est supérieure à 2, le test de sortie de boucle sera satisfait, d'où les résultats constatés.

Il ne faut pas considérer cet exemple comme universel. En effet, dans beaucoup de Basic, vous constaterez que, contrairement à ce qui s'est passé ici, la valeur de 0,2 en mémoire sera légèrement trop petite. Dans ce cas, la boucle se déroulera bien 11 fois avec X valant environ 2 au dernier tour.

REMARQUE IMPORTANTE :

Aucun problème de ce genre n'existe avec les pas entiers, quel que soit le Basic utilisé. Vous pouvez donc toujours éviter les désagréments évoqués ci-dessus en utilisant un compteur à pas entier. Ainsi, l'exemple précédent pourrait s'écrire :

```
10 REM   CALCUL DE X^3 + 4X^2 - 5
20   LET X = 0
30   FOR I = 1 TO 11
40        LET Y = X^3 + 4*X^2 - 5
50        PRINT X,Y
60        LET X = X + .2
70   NEXT I
Ok
RUN
 0             -5
 .2            -4.832
 .4            -4.296
 .6            -3.344
 .8            -1.928
 1              0
 1.2            2.488
 1.4            5.584
 1.6            9.336
 1.8            13.792
 2              19
Ok
```

4.3. Boucles à pas négatif

Vous pouvez également, dans l'instruction FOR, utiliser des pas négatifs, comme dans cet exemple :

```
100 REM   CALCUL DE X^3 + 4X^2 - 5
110    FOR X = 10 TO 0 STEP -1
120       LET Y = X^3 + 4*X^2 - 5
130       PRINT X,Y
140    NEXT X
Ok
RUN
 10            1395
 9             1048
 8             763
 7             534
 6             355
 5             220
 4             123
 3             58
 2             19
 1             0
 0            -5
Ok
```

Ici, le compteur X est initialisé à 10. A chaque tour de boucle, sa valeur est augmentée de – 1, c'est-à-dire diminuée de 1. Le contenu de la boucle (instructions 120 et 130) est répété tant que la valeur du compteur est SUPÉRIEURE ou ÉGALE à la limite. (Notez que le test de sortie de boucle est différent de ce qu'il était avec un pas positif).

REMARQUE :

Lorsque vous utilisez des pas négatifs non entiers, vous risquez de rencontrer des problèmes analogues à ceux dont nous avons parlé pour les pas positifs non entiers.

5.1. Entrée dans une boucle

En principe, Basic interdit «d'entrer» dans une boucle autrement que par l'instruction FOR. Supposons que vous ayez écrit, par maladresse, le programme suivant :

```
100 LET I = 5
110 GOTO 130
120 FOR I = 1 TO 10
130    PRINT I
140 NEXT I
Ok
```

Son exécution vous fournira :

```
RUN
 5
NEXT without FOR in 140
Ok
```

Basic a exécuté l'instruction 100 en affectant la valeur 5 à I. Puis l'instruction 110 l'a amené en 130 qui écrit la valeur de I, c'est-à-dire 5. Basic rencontre alors l'instruction NEXT I alors qu'aucune instruction FOR n'a été rencontrée pour ce compteur I. Il signale cette anomalie par un message qui signifie :

Instruction NEXT sans FOR en 140

5.2. Ne pas créer de boucles inutiles

Exécutons ce programme :

```
100 LET K1 = 50
110 LET K2 = 1
120 FOR K = K1 TO K2
130    PRINT K
140 NEXT K
Ok
RUN
Ok
```

L'instruction 120 affecte la valeur 50 à K. Puis elle compare cette valeur de K à la limite contenue dans K2. Comme 50 est supérieur à 1, Basic considère que la boucle est terminée et il passe à l'instruction suivant le NEXT K. Ici, il rencontre la fin du programme.

Cet exemple vous paraît certainement artificiel car vous ne trouvez aucune raison de programmer ainsi. Toutefois, vous pouvez rencontrer ce phénomène lorsque les bornes (ici K1 et K2) proviennent de calculs dont vous ne connaissez pas d'avance les résultats.

5.3. Ni de boucles infinies

Si nous exécutions ce programme :

```
10 FOR I = 1 TO 4
20      PRINT I
30      LET I = I – 1
40 NEXT I
```

il écrirait indéfiniment la valeur 1. En effet, l'instruction 10 affecte bien la valeur 1 à I. L'instruction 20 écrit donc cette valeur. Par contre, l'instruction 30 affecte à I la valeur 0 tandis que 40 l'augmente de 1, lui redonnant à nouveau la valeur 1. Comme la valeur de I est inférieure à la limite 4, la boucle se poursuit. Le second tour produit exactement le même effet, et ainsi de suite.

D'une manière générale, bien que ce ne soit pas formellement interdit par Basic, nous vous conseillons de ne jamais modifier la valeur du compteur à l'intérieur d'une boucle.

5.4. A chaque FOR son NEXT

Si, par mégarde, vous exécutez ce programme :

```
10 LET N = 5
20 FOR I = 1 TO N
30      PRINT I
40 NEXT N
```

vous obtiendrez :

```
RUN
NEXT without FOR in 40
```

Le message obtenu précise que vous avez voulu incrémenter une variable qui n'était pas, à ce moment là, considérée comme un compteur de boucle.

5.5. Attention à la sortie de boucle

Dans ce programme :

```
100 FOR N = 1 TO 10
110 ...........
120 ...........
130 NEXT N
140 ...........
```

lorsque la boucle a été parcourue 10 fois et que Basic vient exécuter l'instruction 140, le compteur N a pour valeur 11.

De même, dans :

```
100 FOR X = 1 TO 11 STEP 3
110 ...........
120 ...........
130 NEXT X
140 ...........
```

lorsque la boucle a été parcourue 4 fois (X prenant les valeurs 1, 4, 7 et 10) et que l'on exécute l'instruction 140, le compteur X a pour valeur 13.

6. POUR INTERROMPRE L'EXÉCUTION D'UN PROGRAMME

Dès l'instant où vous commencez à utiliser des boucles, il se peut très bien que, par erreur, vous réalisiez une boucle infinie. (Nous en avons d'ailleurs vu quelques possibilités au cours de ce chapitre). Dans ce cas, il est nécessaire de pouvoir interrompre l'exécution du programme. Une manière brutale consiste à mettre votre ordinateur hors tension. Certes, l'objectif est alors atteint, mais vous perdez également votre programme (à moins que vous ne l'ayez préalablement sauvegardé). Vous serez alors obligé d'en taper à nouveau toutes les instructions alors que quelques corrections auraient peut-être suffit.

Une autre solution, plus intéressante, consiste à utiliser une touche spéciale, marquée souvent BREAK ou INTERRUPT ou une combinaison de touches (souvent Control + C). L'exécution est alors interrompue. Basic vous fournit un message précisant quelle instruction il était entrain d'exécuter au moment où vous avez frappé cette touche. Vous vous retrouvez alors en mode normal, c'est-à-dire que vous pouvez alors taper n'importe quelle commande ou, le cas échéant, modifier votre programme.

Vous pouvez, bien sûr, utiliser cette technique pour interrompre n'importe quel programme qui ne se comporte pas comme vous le souhaiteriez. Un avantage particulier de Basic sur la plupart des autres langages est qu'il vous offre alors la possibilité (entre autres choses) d'examiner le contenu de certaines variables de votre programme par des instructions PRINT en mode direct. Vous pouvez même modifier certaines valeurs ou certaines instructions du programme avant d'en poursuivre l'exécution, à partir de l'endroit où il s'était interrompu.

EXERCICES

1. Qu'est-ce qu'un compteur de boucle ?

2. Quelles sont les opérations subies par le compteur d'une boucle ?

3. Quelles sont les deux manières d'écrire une boucle ?

— application : écrire de deux façons différentes, les instructions permettant d'écrire les nombres 4, 5, 6, . . ., 23, 24, 25

4. A combien de tours de boucle correspondent ces instructions FOR et quelles sont les valeurs prises par le compteur (à l'intérieur de la boucle) ?

a) FOR N = 1 TO 4

b) FOR Z = 6 TO 8

c) FOR X = 10 TO 10

5. Quels résultats obtiendra-t-on en exécutant ce programme :

```
10 FOR A = 1 TO 4
20 PRINT "COMPTEUR DANS LA BOUCLE ="; A
30 NEXT A
40 PRINT "COMPTEUR EN SORTIE DE BOUCLE ="; A
```

6. Même question avec :

```
10 FOR A = 1 TO 6 STEP 4
20 PRINT "COMPTEUR DANS LA BOUCLE =" ; A
30 NEXT A
40 PRINT "COMPTEUR EN SORTIE DE BOUCLE =" ; A
```

7. Même question avec :

```
10 LET S = 0
20 FOR I = 1 TO 10 STEP 3
30 LET S = S + I
40 NEXT I
50 PRINT S
```

8. Écrire un programme calculant la somme des carrés des nombres entiers compris entre 1 et 15

9. Même question pour la somme des cubes des nombres impairs compris entre 1 et 17

10. Combien de tours de boucle obtiendra-t-on avec chacune des instructions suivantes et quelles sont les valeurs prises par le compteur ?

a) FOR X = 1 TO 2 STEP 0.1

b) FOR X = 1 TO 2.1 STEP 0.5

c) FOR X = 1 TO 1.4 STEP 0.5

d) FOR X = 5 TO 0 STEP – 1

e) FOR X = 1 TO 0 STEP – 0.1

N.B. Dans certains cas, il y a deux réponses possibles suivant l'ordinateur utilisé.

11. Écrire un programme permettant d'obtenir les valeurs de la fonction :

$$y = \frac{x^2 + 2x + 3}{x + 5}$$

pour x variant de 0 à 2 par pas de 0,05

12. VRAI ou FAUX ? On peut entrer dans une boucle par n'importe quelle instruction.

13. Quelle anomalie comportent les instructions suivantes ?

```
100 FOR N = 1 TO N1
110 PRINT N
120 LET N = N – 1
130 NEXT N
```

14. Même question avec :

```
100 FOR I = 1 TO N
110 PRINT I
120 NEXT N
```

15. Comment interrompre l'exécution d'un programme ?

IX
BOUCLES IMBRIQUÉES

Dans le précédent chapitre, nous avons appris à écrire les instructions FOR et NEXT et nous en avons vu des exemples simples d'utilisation. En fait, le contenu d'une boucle peut être aussi important et varié que vous le souhaitez. En particulier, on pourra très bien y rencontrer des choix réalisés par des instructions IF et GO TO. Qui plus est, on pourra également y trouver d'autres boucles FOR ; dans ce dernier cas, on parlera de «boucles imbriquées».

1. INTRODUCTION

Considérons cette boucle :

```
100 FOR I = 1 TO 3
110      -------
         -------
130      -------
140 NEXT I
```

Nous pouvons très bien placer une nouvelle boucle parmi les instructions 110 à 130. Nous devons seulement prendre soin de lui choisir un compteur différent de I, comme dans cet exemple :

```
100 FOR I = 1 TO 3
110     FOR J = 1 TO 2
120         PRINT I;J
130     NEXT J
140 NEXT I
Ok
```

Exécutons ce petit programme :

```
RUN
 1  1
 1  2
 2  1
 2  2
 3  1
 3  2
Ok
```

Que s'est-il passé ? L'instruction 100 demande de répéter trois fois les instructions 110 à 130, en donnant à I successivement les valeurs 1,2 et 3. Par ailleurs, pour chaque valeur de I, l'instruction 110 demande de répéter deux fois l'instruction PRINT I ; J en donnant à J successivement les valeurs 1 et 2.
L'instruction PRINT I ; J s'est trouvée exécutée 3 x 2 = 6 fois.

2. EXEMPLE D'UTILISATION

Soit à écrire un programme de calcul de la moyenne de chaque élève d'une classe (chaque élève pouvant avoir plusieurs notes). On suppose qu'on entre d'abord toutes les notes du premier élève. Le programme nous en calcule alors la moyenne. Puis, on entre toutes les notes du second élève, et ainsi de suite. Le programme doit également prévoir que le nombre de notes peut différer d'un élève à l'autre.

Si vous considérez uniquement le calcul de la moyenne d'un élève, vous constatez qu'il s'agit d'un problème semblable à celui rencontré dans le paragraphe 2.3 du chapitre précédent. Le programme correspondant pourrait être :

```
100 PRINT "COMBIEN DE NOTES"
110 INPUT N
120 LET S = 0
130 FOR I = 1 TO N
140    PRINT "NOTE"
150    INPUT A
160    LET S = S + A
170 NEXT I
180 LET M = S/N
190 PRINT "MOYENNE =";M
Ok
```

Nous allons maintenant demander que ces instructions soient répétées pour chaque élève. Il faut donc les placer à l'intérieur d'une nouvelle boucle FOR-NEXT dont le compteur peut, par exemple, varier de 1 jusqu'au nombre total d'élèves. Si ce nombre n'est pas connu à l'avance, on peut le faire entrer au début du programme. Ceci nous conduit finalement à écrire :

```
100 PRINT "COMBIEN D'ELEVES"
110 INPUT E
120 FOR J = 1 TO E
130    PRINT "COMBIEN DE NOTES"
140    INPUT N
150    LET S = 0
160    FOR I = 1 TO N
170       PRINT "NOTE"
180       INPUT A
190       LET S = S + A
200    NEXT I
210    M = S/N
220    PRINT "MOYENNE =";M
230 NEXT J
Ok
```

Exécutons ce programme avec un petit nombre d'élèves :

```
RUN
COMBIEN D'ELEVES
? 2
COMBIEN DE NOTES
? 3
NOTE
? 14
```

```
NOTE
? 12
NOTE
? 7.5
MOYENNE = 11.1667
COMBIEN DE NOTES
? 2
NOTE
? 14
NOTE
? 12.5
MOYENNE = 13.25
Ok
```

Vous remarquez que l'utilisation d'un tel programme devient peu aisée dès que le nombre d'élèves est important. On peut améliorer la situation en :

– faisant entrer le nom de l'élève avant d'en demander les notes, et en rappelant son nom lors de l'écriture de sa moyenne.

– donnant un numéro d'ordre à chaque élève.

– donnant un numéro d'ordre à chaque note d'un élève.

Après l'introduction de ces modifications, le programme se présenterait ainsi :

```
100 PRINT "COMBIEN D'ELEVES"
110 INPUT E
120 FOR J = 1 TO E
130    PRINT "NOM DE L'ELEVE NUMERO";J
140    INPUT N$
150    PRINT "COMBIEN DE NOTES POUR L'ELEVE ";N$
160    INPUT N
170    LET S = 0
180    FOR I = 1 TO N
190       PRINT "NOTE NUMERO";I
200       INPUT A
210       LET S = S + A
220    NEXT I
230    M = S/N
240    PRINT "MOYENNE DE L'ELEVE ";N$;" =";M
250 NEXT J
Ok
```

En voici l'exécution, avec les mêmes données que précédemment :

```
RUN
COMBIEN D'ELEVES
? 2
NOM DE L'ELEVE NUMERO 1
? DUPONT
COMBIEN DE NOTES POUR L'ELEVE DUPONT
? 3
NOTE NUMERO 1
? 14
NOTE NUMERO 2
? 12
NOTE NUMERO 3
? 7.5
MOYENNE DE L'ELEVE DUPONT = 11.1667
NOM DE L'ELEVE NUMERO 2
? DUBOIS
COMBIEN DE NOTES POUR L'ELEVE DUBOIS
? 2
NOTE NUMERO 1
? 14
NOTE NUMERO 2
? 12.5
MOYENNE DE L'ELEVE DUBOIS = 13.25
Ok
```

3. QUELQUES RÈGLES CONCERNANT LES BOUCLES IMBRIQUÉES

3.1. Bien imbriquer les boucles

Nous avons présenté les boucles imbriquées, en considérant qu'une boucle était entièrement comprise dans une autre. Par contre si vous écrivez, par mégarde :

```
100 FOR I = 1 TO 10
        ------
150     FOR J = 1 TO 5
            -------
200     NEXT I
        ------
250 NEXT J
```

Vous n'avez plus de boucles imbriquées. Il vous serait d'ailleurs certainement difficile de dire ce que fait un tel programme. Cette situation est, en principe, reconnue par Basic qui vous la signalera par un message d'erreur. Sur notre machine, nous obtenons :

```
NEXT without FOR in 250
```

mais cela peut être différent sur la votre.

3.2. A chaque niveau de boucle son compteur

Si vous écrivez :

```
100 FOR I = 1 TO N
        ------
200     FOR I = 1 TO P
            -------
250     NEXT I
        ------
300 NEXT I
```

vous commettez, en principe, une erreur puisque vous utilisez à nouveau un compteur I, à l'intérieur d'une boucle ayant déjà I comme compteur.

Il est cependant fréquent que Basic ne détecte pas cette anomalie. Dans ce cas, l'instruction 200 va réinitialiser le compteur I à 1. La boucle interne sera parcourue P fois. A sa sortie, J vaudra P + 1. L'instruction 300 affectera alors à I la valeur P + 2. Deux cas sont alors possibles suivant les valeurs relatives de N et P :

a) Si $P + 2 \leqslant N$ la boucle externe sera poursuivie avec $I = P + 2$. Puis l'instruction 200 fera à nouveau s'exécuter la boucle interne. A sa sortie, I vaudra P + 1, etc, . . . On aura ainsi une boucle infinie.

b) Si $P + 2 > N$ la boucle externe ne sera pas poursuivie. Tout se sera passé comme si elle n'existait pas.

EXERCICES

1. Qu'obtiendra-t-on en exécutant ce programme :

```
100 FOR I = 1 TO 3
110       FOR J = 10 TO 20 STEP 10
120             PRINT I ; J
130       NEXT J
140 NEXT I
```

2. Même question avec :

```
100 FOR I = 1 TO 3
110       PRINT I
120 NEXT I
130 FOR J = 10 TO 20 STEP 10
140       PRINT J
150 NEXT J
```

3. Soit le programme :

```
100 FOR X = 1 TO 10
110       PRINT X
120       FOR K = 1 TO 5
130             LET Y = (K + 3) * (X ^2 – 1)
140             PRINT Y
150       NEXT K
160       PRINT "OK"
170 NEXT X
```

Combien de fois exécutera-t-on :

a) l'instruction 110

b) l'instruction 130

c) l'instruction 160

4. Quelle anomalie comporte ce programme :

```
100 FOR Y = 1 TO 3 STEP 0.1
110 PRINT Y
120 FOR X = 1 TO 0 STEP – 0.1
130 LET Z = X ^2 + Y ^2
140 PRINT Z
150 NEXT Y
160 NEXT X
```

5. VRAI ou FAUX ? Deux boucles imbriquées peuvent utiliser le même compteur.

6. Écrire un programme permettant d'écrire toutes les tables de multiplication, de 1 à 9, sous la forme :

TABLE DES 1
1 x 1 = 1
1 x 2 = 2
. . . . .
1 x 9 = 9

TABLE DES 2
2 x 1 = 2
2 x 2 = 4
. . . . .
2 x 9 = 18
etc. . .

7. Écrire un programme calculant :

$$z = 2\,x^2 + 4\,x\,y + 3\,y^2$$

pour tous les couples (x, y) possibles, avec :

x = 0 0,1 0,2 . . . . 0,9 1
y = 0 0,05 0,1 . . . . 0,95 1

X

INSTRUCTIONS READ ET DATA

1. INTRODUCTION

Considérons à nouveau notre programme de calcul des moyennes des élèves d'une classe (deuxième programme du paragraphe 2 du chapitre précédent). A chaque exécution, nous devions entrer au clavier le nombre d'élèves et leurs noms. Ceci était réalisé par les instructions suivantes :

```
110 INPUT E
120 FOR J = 1 TO E
         -----
140      INPUT N$
         -----
         -----
250 NEXT J
```

Si l'on souhaite utiliser plusieurs fois ce programme pour la même classe, on peut trouver regrettable de devoir à chaque fois lui fournir les mêmes informations (nombre et noms des élèves). En fait, Basic vous permet de faire figurer ces informations à l'intérieur même du programme, sous la forme d'instructions DATA. Les instructions INPUT correspondantes sont alors remplacées par des instructions READ. Ces dernières vont directement prélever des valeurs dans les DATA, alors que les premières demandaient d'entrer ces mêmes valeurs au clavier.

Notre programme précédent pourrait être transformé ainsi :

```
100 DATA 10
110 DATA "DUPONT", "DUVAL", "DURAND", "DUBOIS"
120 DATA "DUTRONC", "DULAC", "DUCHENE", "DUMAS"
130 DATA "DUNOYER", "DUMAY"
140 READ E
150 FOR J = 1 TO E
          -----
160       READ N$
          -----
          -----
270 NEXT J
```

La première instruction READ (de numéro 140) accède à la première valeur de la première instruction DATA, c'est-à-dire ici 10. Cette valeur est affectée à la variable E. Tout se passe donc, comme si nous avions répondu 10 à une instruction INPUT E (ou comme si nous avions écrit LET E = 10).

Lorsque Basic exécute l'instruction 160, celle-ci accède à l'information suivante dans les DATA. Comme il n'y a plus de valeurs dans la première instruction DATA (100), Basic prend la première valeur de l'instruction DATA suivante (110). C'est donc la chaîne DUPONT qui est affectée à la variable N$.

Au tour de boucle suivant, cette même instruction READ affectera la chaîne DUVAL à N$. Et ainsi de suite pour les 10 chaînes figurant dans les différents DATA.

Voici le listing complet de ce programme :

```
100 DATA 10
110 DATA "DUPONT","DUVAL","DURAND","DUBOIS"
120 DATA "DUTRONC","DULAC","DUCHENE","DUMAS"
130 DATA "DUNOYER","DUMAY"
140 READ E
150 FOR J = 1 TO E
160    READ N$
170    PRINT "COMBIEN DE NOTES POUR L'ELEVE ";N$
180    INPUT N
190    LET S = 0
200    FOR I = 1 TO N
210       PRINT "NOTE NUMERO";I
220       INPUT A
```

```
230         LET S = S + A
240     NEXT I
250     M = S/N
260     PRINT "MOYENNE DE L'ELEVE ";N$;" =";M
270 NEXT J
Ok
```

Le début de son exécution se présenterait ainsi :

```
RUN
COMBIEN DE NOTES POUR L'ELEVE DUPONT
? 2
NOTE NUMERO 1
? 15
NOTE NUMERO 2
? 12.5
MOYENNE DE L'ELEVE DUPONT = 13.75
COMBIEN DE NOTES POUR L'ELEVE DUVAL
? 4
NOTE NUMERO 1
? 11
NOTE NUMERO 2
? 8.5
```

Bien sûr, il vous est toujours possible de modifier les instructions DATA si vous souhaitez, par exemple, changer les noms des élèves et leur nombre. Dans ce cas, l'utilisation de READ et DATA n'aura d'intérêt que si vous pouvez exécuter plusieurs fois ce même programme, sans avoir à modifier totalement le contenu des DATA.

2. DATA - UNE INSTRUCTION PAS COMME LES AUTRES

Lorsque, pendant l'exécution d'un programme, Basic rencontre une instruction DATA, il ne fait rien et se contente de passer à l'instruction suivante. Ainsi, lorsque nous avons demandé l'exécution du programme précédent, Basic a tout d'abord rencontré l'instruction 100 qui était un DATA. Il s'est contenté de passer à la suivante qui était encore un DATA, et ainsi de suite jusqu'à l'instruction 140.

Nous dirons que DATA est une INSTRUCTION NON EXÉCUTABLE ou encore une DÉCLARATION.

De la même façon, si une instruction IF ou GO TO envoie sur une instruction DATA, il y aura automatiquement passage à l'instruction suivante.

3. COMMENT ORDONNER VOS INSTRUCTIONS DATA

Ces instructions peuvent être placées n'importe où dans votre programme. Elles sont prises en compte par les instructions READ suivant l'ordre croissant des numéros de ligne. Exécutons ce programme :

```
100 FOR I = 1 TO 4
110 READ X
120 DATA 5,-10,20
130 PRINT X
140 DATA 15
150 NEXT I
Ok
RUN
 5
-10
 20
 15
Ok
```

Les valeurs de l'instruction DATA de numéro 120 ont été prises en compte avant celle de l'instruction DATA de numéro 140. Ici, nous aurions pu n'utiliser qu'une seule instruction DATA, regroupant toutes les valeurs, par exemple :

```
120 DATA 5, – 10, 20, 15
```

4. QUAND READ EST TROP GOURMAND

Exécutons ce programme :

```
100 DATA 25,100,-30
110 FOR I = 1 TO 4
120     READ X
130     PRINT X
140 NEXT I
Ok
RUN
 25
 100
-30
Out of DATA in 120
Ok
```

Que se passe-t-il ? Lors du premier tour de boucle, l'instruction READ affecte à X la première valeur de l'instruction DATA, soit 25. Le second tour permet de lui affecter la valeur 100 et le troisième tour la valeur – 30. Par contre, au quatrième tour, lorsqu'il exécute l'instruction READ, Basic découvre qu'il n'y a plus de valeur disponible (puisque l'on est en fin de DATA et qu'il n'y a pas d'autre instruction DATA dans ce programme). Il signale cette anomalie par un message signifiant :

En dehors des données en 120

et il arrête l'exécution du programme.

5. NE PAS CONFONDRE NUMÉRIQUE ET CHAINES

Nous vous avons déjà mis en garde contre les confusions possibles entre le numérique et les chaînes de caractères. Ceci reste valable pour les instructions READ et DATA. Considérez cet exemple :

```
100 DATA "DUPONT", "DURAND"
110 READ E, N$
```

L'instruction READ accède à la première valeur du DATA, c'est-à-dire ici DUPONT qui est une chaîne de caractères. Cette dernière ne peut être affectée à la variable numérique E. Basic signale cette anomalie par un message, maintenant bien connu :

```
Syntax error in 110
```

6. FAUT-IL PLACER LES CHAINES ENTRE GUILLEMETS ?

6.1. Souvent on peut s'en passer

Considérons le cas contraire du précédent :

```
100 DATA 25
110 READ N$
```

L'instruction READ provoque l'affectation à la variable chaîne N$, de la valeur 25. Vous pourriez penser qu'une erreur devrait être détectée par Basic car 25 vous apparaît comme une valeur numérique. Cependant, rien n'empêche de considérer qu'il s'agit d'une chaîne formée des deux caractères 2 et 5. C'est effectivement ce que fera Basic. Bien entendu, il ne sera pas question, ensuite, de faire intervenir la variable N$ dans des calculs.

En fait, cet exemple vous montre que Basic se comporte de la même manière que dans le cas des instructions INPUT mentionnant des variables chaînes. En effet, comme nous l'avons dit dans le chapitre 6, lorsque à cette instruction :

```
110 INPUT N$
```

vous répondez :

```
? 25
```

le résultat est exactement le même que si vous répondiez :

```
" 25 "
```

De la même façon, nous aurions pu omettre les guillemets de tous les DATA de notre exemple d'introduction. Par exemple, l'instruction 110 aurait pu s'écrire :

```
110 DATA DUPONT, DUVAL, DURAND, DUBOIS
```

REMARQUE IMPORTANTE

Ce que nous avons dit à propos des chaînes dans les DATA s'applique aussi, comme nous l'avons signalé, aux réponses à l'instruction INPUT. Par contre, lorsqu'une constante chaîne apparaît dans une instruction d'affectation elle doit TOUJOURS ETRE PLACÉE ENTRE GUILLEMETS.

6.2. Mais parfois c'est indispensable

6.2.1. Pour obtenir des blancs

Exécutons ce programme :

```
100 DATA CECI,EST,UN,EXEMPLE
110 READ A$,B$,C$,D$
120 PRINT A$;B$;C$;D$
Ok
RUN
CECIESTUNEXEMPLE
Ok
```

L'absence de blancs entre les mots s'explique par le fait que nous n'en avons pas prévus dans les 4 chaînes fournies dans l'instruction DATA. Nous pensons alors à ajouter quelques blancs, en procédant comme ceci :

```
100 DATA CECI ,EST ,UN ,EXEMPLE
110 READ A$,B$,C$,D$
120 PRINT A$;B$;C$;D$
Ok
RUN
CECIESTUNEXEMPLE
Ok
```

A notre surprise, le résultat est inchangé. Que s'est-il passé ? En fait, nous l'avons déjà constaté, les caractères blancs ne sont pas significatifs pour Basic. Nous pouvons en ajouter à volonté dans les instructions pour qu'elles nous paraissent plus lisibles. Nous l'avons déjà fait pour mettre en évidence le contenu d'une boucle en ajoutant plusieurs blancs devant les instructions situées à l'intérieur.

Il se trouve que cette facilité devient un inconvénient, dans le cas de notre dernier exemple. Il est toutefois facile d'y remédier en plaçant entre guillemets les chaînes concernées, ce qui nous conduit au programme :

```
100 DATA "CECI ","EST ","UN ","EXEMPLE"
110 READ A$,B$,C$,D$
120 PRINT A$;B$;C$;D$
Ok
RUN
CECI EST UN EXEMPLE
Ok
```

6.2.2. Pour obtenir des virgules

Si l'on souhaite placer une virgule dans une chaîne de caractères, il est également nécessaire d'utiliser des guillemets pour éviter qu'elle ne soit considérée comme un séparateur de deux valeurs.

Ainsi, si nous exécutons ce programme :

```
100 DATA D'ACCORD, CA MARCHE
110 READ M$
120 PRINT M$
Ok
RUN
D'ACCORD
Ok
```

l'instruction READ affecte à M$ la première valeur du DATA, c'est-à-dire la chaîne D'ACCORD.

Par contre :

```
100 DATA "D'ACCORD, CA MARCHE"
110 READ M$
120 PRINT M$
Ok
RUN
D'ACCORD, CA MARCHE
Ok
```

fournit bien le résultat escompté.

7. POUR LIRE PLUSIEURS FOIS LES MÊMES DATA_RESTORE

Considérez à nouveau le programme du paragraphe 1 qui vous permettait de calculer la moyenne des élèves d'une classe. Vous pouvez l'utiliser pour calculer les moyennes d'une classe dans une matière, par exemple en mathématiques. Pour calculer les moyennes de la même classe en français, il vous faut alors exécuter à nouveau ce même programme. Vous pouvez alors penser à le modifier pour qu'il calcule les moyennes d'une même classe dans plusieurs matières différentes. Pour cela, vous voyez qu'il suffit de répéter le traitement précédent autant de fois qu'il y a de matières (ce nombre pouvant être fournit par INPUT). Une difficulté apparaît alors, liée au fait que pour traiter la seconde matière, vous devez, à nouveau, relire les noms des élèves. Ceci paraît impossible, puisque l'on est en fin de DATA. Il faudrait pouvoir demander à Basic de reprendre l'examen des instructions DATA depuis le début. C'est effectivement le rôle de l'instruction RESTORE.

Le programme ainsi modifié pourrait être :

```
100 DATA 10
110 DATA "DUPONT","DUVAL","DURAND","DUBOIS"
120 DATA "DUTRONC","DULAC","DUCHENE","DUMAS"
130 DATA "DUNOYER","DUMAY"
132 PRINT "COMBIEN DE MATIERES"
135 INPUT M
137 FOR K = 1 TO M
138    PRINT "MATIERE NUMERO";K
140    READ E
150    FOR J = 1 TO E
160       READ N$
170       PRINT "COMBIEN DE NOTES POUR L'ELEVE ";N$
180       INPUT N
190       LET S = 0
200       FOR I = 1 TO N
210          PRINT "NOTE NUMERO";I
220          INPUT A
230          LET S = S + A
240       NEXT I
250       M = S/N
260       PRINT "MOYENNE DE L'ELEVE ";N$;" =";M
270    NEXT J
275    RESTORE
280 NEXT K
Ok
```

Après avoir effectué les moyennes des 10 élèves dans la première matière, Basic rencontre l'instruction RESTORE (en 275) qui signifie que la prochaine instruction READ (si elle existe) recommencera à examiner les DATA depuis le début. Ainsi, 280 NEXT K nous fait passer à la valeur suivante de K, soit 2. L'instruction 140 permet donc d'affecter à nouveau la valeur 10 à E. L'instruction 160 affecte à N$ le nom du premier élève, puis du second, et ainsi de suite.

EXERCICES

1. VRAI ou FAUX ?

a) L'instruction DATA permet d'éviter l'introduction au clavier de valeurs qui sont toujours les mêmes.

b) L'instruction DATA est toujours associée à l'instruction INPUT.

c) L'instruction PRINT est toujours associée à l'instruction DATA

d) L'instruction READ est toujours associée à l'instruction DATA.

e) On ne peut placer qu'une instruction DATA dans un programme.

f) Les instructions DATA doivent obligatoirement se suivre.

g) L'instruction DATA est une instruction exécutable au même titre que INPUT ou PRINT ou LET.

2. Que produira l'exécution de ce programme :

```
100 DATA 1, 5, 10
110 READ A
120 PRINT A
130 READ B, C
140 PRINT B ; C
```

3. Même question avec :

```
100 DATA 1,5
110 READ A
120 PRINT A
130 READ B, C
140 DATA 10
150 PRINT B ; C
```

4. Même question avec :

```
100 DATA 1, 5
110 READ A, B
```

```
120 PRINT A ; B
130 READ C
140 PRINT C
```

5. Même question avec :

```
100 READ A
110 DATA 10, 50
120 READ B, C
130 DATA 80
140 PRINT A, B, C
150 PRINT A ; B ; C
```

6. Même question avec :

```
100 READ N$
110 READ A, B$
120 DATA DUPONT, 15
130 DATA "ANS"
140 PRINT N$, "A", A, B$
150 PRINT N$ ;  "A" ; A ; B$
```

7. Quand les guillemets sont-ils indispensables dans les instructions DATA ?

8. Quelle différence y-a-t-il entre ces deux programmes ?

```
100 DATA 128          100 DATA 128
110 READ N            110 READ N$
```

9. Que se passe-t-il si l'on exécute ce programme :

```
100 DATA 12
110 READ N$
120 LET N$ = N$ + 1
130 PRINT N$
```

10. Comparez les résultats de ces deux programmes :

```
100 DATA 155                     100 DATA 155
110 READ N                       110 READ N$
120 PRINT "VALEUR =" ; N         120 PRINT "VALEUR =" ; N$
```

11. Complétez le programme suivant pour qu'il fournisse les résultats proposés.

```
100 DATA DUPONT, PARIS
110 READ N$, V$
120 . . . . . . . . . .
RUN
MONSIEUR DUPONT HABITE A PARIS
```

12. Est-il possible de compléter le programme suivant pour qu'il fournisse les résultats proposés ? Pourquoi ?

```
100 DATA 16
110 READ N
120 PRINT N . . . . . . . .
RUN
16EME
```

13. A quoi sert l'instruction RESTORE ?

14. Quels seront les résultats obtenus lors de l'exécution de ce programme ?

```
100 DATA 10, 20, 30, 40
110 READ A, B
120 PRINT A ; B
130 RESTORE
140 FOR I = 1 TO 4
150       READ X
160       PRINT X
170 NEXT I
```

15. Même question avec :

```
100 DATA «XXXXX», «XXX», «X»
110 READ L1$, L2$, L3$
120 PRINT L1$
130 FOR I = 1 TO 5
140       IF I = 3 THEN 170
150             PRINT L3$
160             GO TO 190
170       REM
180             PRINT L2$
190 NEXT I
200 PRINT L1$
```

XI
TABLEAUX À UN INDICE

1. INTRODUCTION

Supposez que l'on souhaite écrire un programme qui demande d'entrer quatre nombres et qui, pour chacun d'entre-eux, en écrit le carré et le cube. On pourrait procéder comme ceci :

```
100 FOR I = 1 TO 4
110     PRINT "VALEUR"
120     INPUT X
130     PRINT X^2,X^3
140 NEXT I
Ok
```

L'exécution se présenterait ainsi :

```
RUN
VALEUR
? 3
 9              27
VALEUR
? 5
 25             125
VALEUR
? 11
 121            1331
VALEUR
? 9
 81             729
Ok
```

On peut trouver cette présentation peu claire et désirer que les résultats soient regroupés, par exemple comme ceci :

```
RUN
VALEUR
? 3
VALEUR
? 5
VALEUR
? 11
VALEUR
? 9
NOMBRE          CARRE          CUBE
 3              9               27
 5              25              125
 11             121             1331
 9              81              729
Ok
```

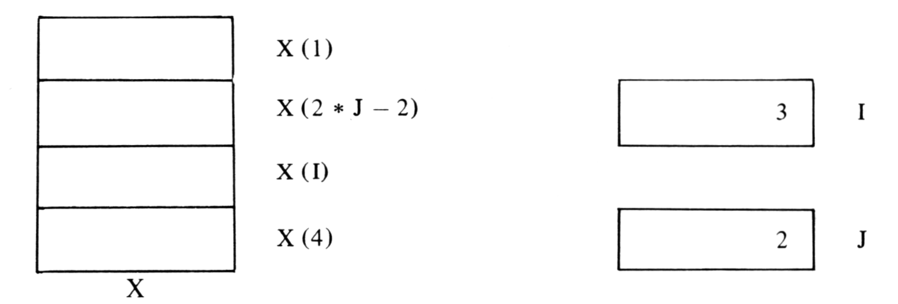
Dans ce cas, vous voyez que le programme doit conserver les quatre valeurs entrées avant de commencer à écrire les résultats correspondant à la première. Dans ces conditions, il n'est plus possible d'utiliser une seule variable X à laquelle on affecte successivement chaque valeur entrée.

Peut-être pensez-vous qu'il faut alors utiliser quatre variables différentes. Ce serait, bien sûr, possible mais nous aurions alors besoin d'écrire une instruction INPUT et une instruction PRINT pour chacune d'entre-elles.

En fait, Basic nous offre une solution plus simple. Elle consiste à choisir un nom unique (par exemple X) pour l'ensemble de ces quatre valeurs. X s'appelle alors un «tableau». Comme X désigne, à lui seul, quatre éléments différents, il nous faut «repérer» chaque élément de X. Cela se fait à l'aide de ce que l'on appelle un «indice», c'est-à-dire une valeur numérique précisant l'emplacement d'un élément dans le tableau. Ainsi, le premier élément de X est désigné par X (1), le quatrième par X (4).

Cet indice peut aussi être une variable. Ainsi, si I contient la valeur 3, X (I) désigne le troisième élément de X. Plus généralement, on peut également utiliser n'importe quelle expression arithmétique. Si J contient la valeur 2, X (2 * J – 2) désigne le deu-

xième élément de X. Nous pouvons «illustrer» cela par le schéma suivant :



Voyons maintenant ce que serait le programme permettant de résoudre notre problème, c'est-à-dire conserver dans un tableau nommé X les quatre valeurs entrées au clavier, avant d'en calculer et d'en écrire les carrés et les cubes.

```
100     DIM X(4)
110 REM    ---- ENTREE DES 4 VALEURS
120     FOR I = 1 TO 4
130        PRINT "VALEUR"
140        INPUT X(I)
150     NEXT I
160 REM    ---- ECRITURE DES RESULTATS
170     PRINT "NOMBRE","CARRE","CUBE"
180     FOR I = 1 TO 4
190        PRINT X(I),X(I)^2,X(I)^3
200     NEXT I
Ok
```

L'instruction 100 demande à Basic de réserver quatre emplacements pour un tableau nommé X. Les instructions 120 à 150 permettent d'entrer les quatre valeurs de X. En effet, au premier tour de boucle, I vaut 1 et donc INPUT X (I) affecte la valeur entrée au clavier à X (1). Au second tour de boucle, il en va de même pour X (2) et ainsi de suite. De la même manière, les instructions 170 à 200 écrivent chacune des quatre valeurs de X, suivie de son carré et de son cube.

2. COMMENT CRÉER ET UTILISER UN TABLEAU

2.1. Choix du nom du tableau

Vous pouvez fabriquer des noms de tableau suivant les mêmes règles que celles utilisées pour les noms de variable. Un nom de tableau est donc formé d'une lettre suivie éventuellement d'un chiffre. En outre le caractère $ doit être placé à la fin du nom s'il s'agit d'un tableau de chaînes de caractères.

2.2. L'instruction DIM

Lorsque vous utilisez un tableau, il est nécessaire que Basic en connaisse la taille pour pouvoir lui attribuer des emplacements en mémoire. C'est le rôle de l'instruction DIM de préciser quelle est la taille que vous souhaitez donner à un tableau.

Ainsi :

```
DIM A (25)
```

demande de réserver 25 emplacements pour un tableau nommé A. On dit aussi qu'elle réserve un tableau de dimension 25.

De même :

```
DIM C (100), Z$ (20)
```

réserve 100 emplacements pour un tableau numérique C et 20 pour un tableau de chaînes nommé Z$.

2.3. Comment introduire des valeurs dans un tableau

Supposez que vous ayez réservé un tableau C par :

```
DIM C (4)
```

Basic ne possède pas d'instruction permettant de «manipuler» simultanément tous les éléments de C. Il vous faut donc considérer ce tableau comme constitué de quatre variables C (1), C (2), C (3) et C (4). Celles-ci sont nommées «variables indicées» afin de les distinguer des «variables simples» que nous avions rencontrées jusqu'ici.

Ainsi, l'instruction :

```
100 LET C (1) = 10
```

vous permet de ranger la valeur 10 dans le premier élément de C. Si l'on souhaite affecter respectivement les valeurs 20, 25 et 12 aux éléments suivants, on écrira :

```
110 LET C (2) = 20
120 LET C (3) = 25
130 LET C (4) = 12
```

Vous pouvez donc utiliser l'instruction LET pour affecter des valeurs à des variables indicées.

De la même façon, vous pouvez mentionner des variables indicées dans les instructions INPUT ou READ. Ainsi, les éléments de C se verraient affecter les mêmes valeurs que précédemment par :

```
100 DATA 10, 20, 25, 12
110 FOR I = 1 TO 4
120       READ C (I)
130 NEXT I
```

Enfin, nous pouvons affecter à C, des valeurs lues au clavier par :

```
100 INPUT C (1)
110 INPUT C (2)
120 INPUT C (3)
130 INPUT C (4)
```

ou encore :

```
100 FOR I = 1 TO 4
110       INPUT C (I)
120 NEXT I
```

2.4. Comment utiliser les valeurs contenues dans un tableau

Là encore, vous pouvez utiliser un élément de tableau (variable indicée) comme n'importe quelle variable simple.

Ainsi, concernant le tableau C précédent, nous pouvons écrire :

```
LET A = C (1) + 4
```

ou encore :

```
LET Y = C (3) + C (4) ^2
```

ou encore :

```
LET Z = C (K) ^2 + 3 * C (I)
```

(à condition que I et K contiennent des valeurs comprises entre 1 et 4)

Cette instruction :

```
LET C (4) = C (1) + C (2)
```

affectera au quatrième élément de C, la somme des deux premiers éléments.

De même :

```
PRINT C (1)
```

permet d'écrire la valeur du premier élément de C.

Enfin :

```
PRINT C (I) ^2
```

écrit, si I contient une valeur comprise entre 1 et 4, le carré de l'élément correspondant de C.

3. QUAND LA DIMENSION EST VARIABLE

Il se peut que vous ne sachiez pas quelle taille donner à certains tableaux au moment où vous écrivez le programme. Vous pouvez alors placer, dans votre instruction DIM, une expression arithmétique dont la valeur représente la taille à attribuer au tableau. En voici deux exemples :

```
DIM T (N)
DIM X (2 * N + 1)
```

Ainsi, ces instructions :

```
100 PRINT "COMBIEN DE NOMBRES"
110 INPUT N
120 DIM X (N)
130 FOR K = 1 TO N
140     PRINT "VALEUR"
150     INPUT X (K)
160 NEXT K
```

constituent un tableau de dimension quelconque, à partir de valeurs entrées au clavier.

4. UN EXEMPLE : TRI ALPHABÉTIQUE

Nous voulons réaliser un programme qui lit un nombre quelconque de noms (donc de chaînes de caractères) et qui les réécrit, rangés par ordre alphabétique. Les notions que nous allons utiliser sont :

– tableaux de chaînes de caractères

– dimensions variables

– la comparaison des variables chaînes permet de les ranger par ordre alphabétique.

La méthode de tri proposée est la suivante : nous considérons le premier nom (donc le premier élément du tableau) que nous comparons au second. Si ce premier nom vient avant le second, par ordre alphabétique, nous ne faisons rien. Dans le cas contraire, nous échangeons les contenus des deux variables correspondantes (c'est-à-dire que le second nom prend la place du premier et réciproquement). Nous comparons alors le premier nom (dont la valeur peut éventuellement avoir changé) avec le troisième, en échangeant leurs valeurs s'il y a lieu, et ainsi de suite jusqu'au dernier nom. A ce moment là, nous sommes alors certains que le premier nom est bien le premier dans l'ordre alphabétique de tous les noms considérés. Nous sommes ainsi ramenés au problème initial mais, cette fois, il porte sur un élément de moins. Nous recommençons donc une série de comparaisons entre le deuxième nom et tous ses suivants, ce qui nous fera apparaître le second nom par ordre alphabétique en deuxième position. Et ainsi de suite jusqu'à l'avant dernier nom.

Voici le programme correspondant et un exemple d'exécution.

```
100 REM      TRI ALPHABETQUE DE NOMS
110 REM
120    PRINT "COMBIEN DE NOMS A TRIER"
130    INPUT N
140 REM
150    DIM T$(N)
160 REM
170 REM   ---- ENTREE DES NOMS
180       PRINT "ENTREZ VOS NOMS A TRIER"
190       FOR I = 1 TO N
200          INPUT T$(I)
210       NEXT I
220 REM
230 REM   ---- TRI DES NOMS
240 REM   ---- BOUCLE SUR LES NOMS 1 A N-1
250       FOR I = 1 TO N-1
260 REM        ---- COMPARAISON NOM DE RANG I A TOUS SES SUIVANTS
270          FOR J = I+1 TO N
280             IF T$(I) < T$(J) THEN 320
290             LET X$ = T$(I)
300             LET T$(I) = T$(J)
310             LET T$(J) = X$
320          NEXT J
330       NEXT I
340 REM
350 REM   ---- ECRITURE DES NOMS TRIES
360       PRINT "VOICI VOS NOMS TRIES"
370       FOR I = 1 TO N
380          PRINT T$(I)
390       NEXT I
Ok
RUN
COMBIEN DE NOMS A TRIER
? 5
ENTREZ VOS NOMS A TRIER
? TABLE
? CHAISE
? PARIS1
? TABLEAU
? PARIS2
VOICI VOS NOMS TRIES
CHAISE
PARIS1
PARIS2
TABLE
TABLEAU
Ok
```

5. PRÉCAUTIONS

Cette nouvelle notion qu'est le tableau vous permet donc de manipuler facilement une grande quantité d'informations de même nature. Ainsi, les simples instructions :

```
100 DIM C (1000)
110 FOR I = 1 TO 1000
120      LET C (I) = I ^2
130 NEXT I
```

vous permettent-elles de créer un tableau contenant les carrés des 1000 premiers nombres entiers.

Cependant cette facilité nouvelle est assortie de risques tout aussi nouveaux pour vous. Nous allons tenter de vous les faire découvrir.

5.1. Bien placer l'instruction DIM

C'est seulement lorsque Basic exécute cette instruction qu'il réserve la place pour le tableau correspondant. Ce qui signifie qu'il ne suffit pas d'avoir placé une instruction DIM dans son programme pour qu'il soit correct. Encore faut-il, en outre, qu'elle ait été exécutée avant toute utilisation du tableau correspondant.

Ainsi, dans ce programme :

```
100 INPUT T
110 DIM A (20)
120 FOR I = 1 TO 20
130      LET A (I) = (1 + T) ^I
140 NEXT I
```

l'instruction DIM est bien exécutée avant l'utilisation de A (en 130).

Par contre si, par maladresse, vous écrivez un programme possédant des branchements (GO TO) intempestifs, il se peut que ces derniers vous masquent une mauvaise disposition d'une instruction DIM. Considérez cet exemple :

```
100 REM  PROGRAMME BIZARD
110     GOTO 170
120     DIM Y(15)
130     FOR K = 1 TO 15
140        PRINT Y(K)
150     NEXT K
160     GOTO 220
170     FOR K = 1 TO 15
180        READ Y(K)
190     NEXT K
200     GOTO 120
210     DATA 1,2,8,4,-3,5,-8,7,-11,2,1,4,8,4,1
220 REM
Ok
```

Bien qu'apparaissant en début de programme, l'instruction DIM n'a pas encore été exécutée lorsque Basic arrive sur les instructions 170 à 190 qui utilisent le tableau Y. Une tentative d'exécution de ce programme vous fournirait ce message :

Subscript out of range in 180

qui se traduit par :

Indice en dehors des limites en 180

Ceci signifie qu'à un moment donné, dans l'instruction 180, la valeur de K est sortie des limites permises pour les indices du tableau Y. Si l'instruction 120 avait été exécutée, ces limites seraient 1 et 15 et le programme se déroulerait correctement. Par contre, ici, cette instruction n'a pas encore été rencontrée par Basic qui n'a donc pas pu attribuer d'emplacement au tableau Y.

Cependant, vous constatez que le message obtenu ne dit pas exactement que Y n'a pas reçu d'emplacement, mais seulement que l'indice utilisé sort des limites autorisées. L'explication réside dans le fait que lorsque Basic rencontre un nom de tableau n'ayant pas reçu de taille, il lui en attribue une d'office. Généralement, la taille standard ainsi définie est de 10 (mais ce nombre peut être différent dans votre système).

Ainsi, dans notre programme «bizard», lorsque Basic a rencontré l'instruction READ Y (K) pour la première fois, il a réservé un tableau Y de dimension 10. Ce n'est que lorsque K a atteint la valeur 11 que Basic a signalé que cette dernière était en dehors des limites permises.

Vous pouvez, si vous le souhaitez, confirmer cette explication en ajoutant au programme, l'instruction :

```
175 PRINT K
```

Vous constaterez alors que le programme se déroule correctement pour K allant de 1 à 10, et ce n'est que pour K = 11 qu'un message d'erreur est imprimé.

De même, ce programme s'exécute correctement :

```
100 DATA 25,15,4,8
110 FOR I = 1 TO 4
120     READ A(I)
130 NEXT I
140 FOR I = 1 TO 4
150     PRINT A(I)
160 NEXT I
Ok
RUN
 25
 15
 4
 8
Ok
```

malgré l'absence d'instruction DIM pour le tableau A.

REMARQUE :

Si vous souhaitez écrire des programmes lisibles et facilement compréhensibles, nous vous conseillons d'utiliser systématiquement des instructions DIM pour tous les tableaux utilisés, même si leur dimension est inférieure à 10.

5.2. Quand la dimension est variable

5.2.1. S'assurer qu'elle a la bonne valeur

Nous avons vu qu'on pouvait exprimer la taille d'un tableau sous la forme d'une expression arithmétique. Dans ce cas, il faut s'assurer que cette expression est bien «définie» au moment voulu, autrement dit qu'elle ne fait pas intervenir des variables non définies (ou non encore définies).

Exécutons ce programme :

```
100 DIM A(2*N)
110 INPUT N
120 FOR I = 1 TO 2*N
130     PRINT "ELEMENT NO";I
140     INPUT A(I)
150 NEXT I
Ok
RUN
? 10
ELEMENT NO 1
? 45
Subscript out of range in 140
Ok
```

Que se passe-t-il ? Quand Basic exécute l'instruction 100, la variable N n'est pas encore définie. Comme nous l'avons dit au chapitre 5, il est fréquent que, dans ce cas, on y trouve la valeur 0 (C'est ce qui se passe dans le Basic que nous utilisons). Dans ces conditions, 2 * N prend également la valeur 0. L'instruction 100 réserve donc un tableau de dimension 0. Ceci explique qu'en 140, Basic ne puisse accepter de ranger une valeur dans le premier élément de ce tableau, d'où le message obtenu.

Ici, il est facile de remédier à cette erreur en changeant l'ordre des deux premières instructions :

```
100 INPUT N
110 DIM A (2 * N)
```

REMARQUE :

Nous avons dit qu'en l'absence de déclaration de dimension, Basic prévoyait 10 emplacements pour un tableau. Par contre, dès qu'une instruction DIM a été rencontrée, la valeur prise en compte est celle qui y est mentionnée, même si elle est inférieure à 10. C'est ce qui s'est passé dans notre dernier exemple.

5.2.2. Quand elle n'est pas entière

Dans ce cas, l'expression est arrondie à l'ENTIER LE PLUS PROCHE. Vérifions-le sur un exemple :

```
100 INPUT N
110 DIM T(N)
120 FOR I = 1 TO N+2
130    LET T(I) = I
140    PRINT T(I)
150 NEXT I
Ok
RUN
? 2.3
 1
 2
Subscript out of range in 130
Ok
RUN
? 2.6
 1
 2
 3
Subscript out of range in 130
Ok
```

Dans la première exécution, la dimension attribuée par Basic a été de 2 (pour N = 2.3) puisque le message d'erreur a été obtenu pour I = 3. Dans le second cas la dimension retenue a été de 3 pour N = 2.6.

5.3. Ne pas changer d'avis en cours de programme

Autrement dit, ne pas donner deux instructions DIM pour le même tableau. Exécutons ce programme :

```
100 DIM A(10)
110 FOR K = 1 TO 10
120    LET A(K) = K
130 NEXT K
140 DIM A(5)
150 FOR J = 1 TO 5
160    LET A(J) = J
170 NEXT J
Ok
RUN
Duplicate Definition in 140
Ok
```

En nous écrivant un message se traduisant par :

Définition apparaissant en double en 140

Basic nous signale qu'en 140, nous avons voulu redéfinir la dimension du tableau A.

EXERCICES

1. Qu'est-ce-qu'un tableau ?

2. Comment s'écrivent les noms de tableau en Basic ?

3. Qu'est-ce-qu'un indice ?

4. Qu'est-ce-qu'une variable indicée ?

5. Comment peut-on écrire les indices en Basic ?

6. A quoi sert l'instruction DIM ?

7. VRAI ou FAUX ?

a) Les tableaux Basic ne peuvent contenir que des valeurs numériques.

b) Les indices peuvent prendre n'importe quelle valeur.

c) L'instruction DIM doit être placée en début de programme.

d) L'emplacement de l'instruction DIM n'a aucune importance.

e) Utiliser un tableau permet de donner un seul nom à un ensemble de valeurs.

f) Un tableau correspond toujours à une variable indicée.

g) Un tableau correspond généralement à plusieurs variables indicées.

8. Que font ces instructions :

a) DIM C (50), M$ (25)

b) DIM C3 (25), X8 (4), Z5$ (15)

9. Que font ces instructions :

```
100 DIM H (4)
110 LET H (3) = 8
120 LET H (2) = 6
130 LET H (4) = 9
140 LET H (1) = 3
```

10. Même question avec

```
100 DIM K1 (5)
110 FOR I = 1 TO 5
120       LET K1 (I) = 2 * I - 1
130 NEXT I
```

11. Même question avec :

```
100 DIM C (12)
110 FOR K = 1 TO 6
120       LET C (2 * K) = 0
130       LET C (2 * K - 1) = 1
140 NEXT K
```

12. Même question avec :

```
100 DIM Z9 (10)
110 FOR K = 1 TO 4
120       LET Z9 (K) = K
130 NEXT K
140 LET Z9 (5) = 99
150 FOR K = 6 TO 10
160       LET Z9 (K) = K - 5
170 NEXT K
```

13. Écrivez des instructions permettant d'écrire les valeurs du tableau Z9 de l'exercice précédent.

14. Soit V un tableau de 1000 éléments. Ecrivez les instructions permettant de calculer dans une variable S la somme de tous ses éléments.

15. Quelle sera la taille attribuée par Basic au tableau C, dans chacun des cas suivants :

a)
```
100 DIM C (50)
```

b)
```
100 LET N = 5
110 DIM C (N ^2 - 3)
```

c)
```
100 FOR I = 1 TO 5
110 LET C (I) = I ^2
120 NEXT I
```

16. Quelle sera la taille attribuée au tableau Y si l'on exécute ce programme :

```
100 INPUT X
110 DIM Y (2 * X + 1)
. . . . . . . . . .
```

en entrant comme valeur pour X :

a) 10

b) 12.2

c) 12.3

d) 12.6

e) 12.8

17. Que va-t-il se passer si l'on exécute ce programme :

```
100 DIM T (5)
110 INPUT N
120 FOR K = 1 TO N + 1
130         LET T (K) = K ^2
140         PRINT T (K)
150 NEXT K
```

en répondant pour N :

a) 3

b) 5

XII
INSTRUCTION ON GO TO

1. INTRODUCTION

Nous avons appris, dans le chapitre 7, comment programmer un choix entre deux possibilités. Il se peut que vous ayez besoin de réaliser un choix entre plus de deux possibilités. C'est par exemple le cas si vous écrivez un programme permettant de jouer à quatre jeux différents, où l'utilisateur peut choisir l'un des jeux par un nombre compris entre 1 et 4. Le programme pourrait se présenter ainsi :

```
100 PRINT "QUEL JEU CHOISISSEZ VOUS (1 A 4)"
110 INPUT N
120 IF I = 1 THEN 170
130 IF I = 2 THEN 300
140 IF I = 3 THEN 400
150 IF I = 4 THEN 500
160 END
170 REM JEU NUMERO 1
          ------
          ------
300 REM JEU NUMERO 2
          ------
          ------
400 REM JEU NUMERO 3
```

```
------
------
500 REM JEU NUMERO 4
------
------
```

Bien sûr, nous n'avons pas précisé quelles seraient les instructions correspondant à chaque jeu.

Basic va vous permettre de simplifier l'écriture des instructions 120 à 150. En effet, vous pouvez les remplacer par :

```
120 ON I GO TO 170, 300, 400, 500
```

qui précise à quelle instruction on souhaite aller en fonction de la valeur de I.

2. L'INSTRUCTION ON GO TO

```
120 ON I GO TO 170, 300, 400, 500
```

demande à Basic de poursuivre l'exécution :

* à l'instruction de numéro 170 si I vaut 1
* à l'instruction de numéro 300 si I vaut 2
* à l'instruction de numéro 400 si I vaut 3
* à l'instruction de numéro 500 si I vaut 4

De même :

```
100 ON K – 1 GO TO 300, 350, 640
```

demandera à Basic de poursuivre l'exécution :

* en 300 si K – 1 vaut 1 (c'est-à-dire si K vaut 2)
* en 350 si K – 1 vaut 2 (c'est-à-dire si K vaut 3)
* en 640 si K – 1 vaut 3 (c'est-à-dire si K vaut 4)

D'une manière générale, vous pouvez placer après les mots GO TO, autant de numéros de ligne que vous le souhaitez.

Vous pouvez, si c'est nécessaire, utiliser plusieurs fois le même numéro de ligne. Ainsi :

```
ON A GO TO 300, 450, 300, 600, 450
```

poursuivra l'exécution :

* en 300 si A vaut 1 ou 3
* en 450 si A vaut 2 ou 5
* en 600 si A vaut 4

3. EXEMPLE D'UTILISATION : CALCUL DE TVA

Voici un petit programme calculant le prix t.t.c. d'un produit lorsque l'on lui fournit :

– le prix hors taxe

– le type de TVA:

* 1 correspond à une TVA de 7 %
* 2 correspond à une TVA de 12,32 %
* 3 correspond à une TVA de 17,6 %
* 4 correspond à une TVA de 33,33 %

```
100 REM   CALCUL D'UN PRIX TTC
110 REM
120     PRINT "ENTRER LE PRIX HORS TAXE"
130     INPUT P1
140     PRINT "ENTRER LE CODE TVA (1 A 4)"
150     INPUT K
160     ON K GOTO 170,190,210,230
170        LET P2 = P1*1.07
180     GOTO 240
190        LET P2 = P1*1.1232
200     GOTO 240
210        LET P2 = P1*1.176
220     GOTO 240
230        LET P2 = P1*1.3333
240     REM
250     PRINT "PRIX TTC=";P2
Ok
```

Exécutons le dans deux cas :

```
RUN
ENTRER LE PRIX HORS TAXE
? 100
ENTRER LE CODE TVA (1 A 4)
? 1
PRIX TTC= 107
Ok
RUN
ENTRER LE PRIX HORS TAXE
? 750
ENTRER LE CODE TVA (1 A 4)
? 3
PRIX TTC= 882
Ok
```

4. PRÉCAUTIONS

4.1. Quand l'expression suivant ON n'est pas entière

Basic en utilise alors la valeur entière la plus proche. Vérifions-le sur ce programme :

```
100 INPUT X
110 ON X GOTO 120,140
120     PRINT "TOUT SE PASSE COMME SI X = 1"
130 GOTO 150
140     PRINT "TOUT SE PASSE COMME SI X = 2"
150 REM
Ok
RUN
? 1.3
TOUT SE PASSE COMME SI X = 1
Ok
RUN
? 1.6
TOUT SE PASSE COMME SI X = 2
Ok
```

Vous voyez que pour X = 1,3 Basic a utilisé le premier numéro de ligne de l'instruction ON GO TO. Par contre, pour X = 1,6 c'est le deuxième numéro qui a été pris en compte.

4.2. Quand l'expression suivant ON est trop petite ou trop grande

Examinez ce petit programme :

```
100 INPUT K
110 ON K GO TO 120, 140
120 PRINT «CAS1»
130 GO TO 150
140 PRINT «CAS2»
150 REM
```

Tant que nous fournissons pour K une valeur ayant comme partie entière 1 ou 2, il n'y a aucun problème. Par contre, que se passera-t-il si vous frappez 0 ou 4 ?

En fait, la réponse à cette question dépend du système utilisé. Certains Basic vous fourniront un message d'erreur précisant que la valeur de K est en dehors des limites permises. D'autres, au contraire, ne vous donneront aucun message et se contenteront d'exécuter l'instruction suivant le ON GO TO. Ici, ce serait celle de numéro 120. Si c'est de cette manière que fonctionne votre Basic (et vous vous en apercevrez aisément en essayant le programme ci-dessus), nous vous conseillons vivement d'ajouter avant ON GO TO des instructions vérifiant que l'expression utilisée a une valeur correcte.

5. LIMITES DU ON GO TO

La nature même de cette instruction impose que le choix soit basé sur la valeur d'une expression. Très souvent, vous rencontrerez des choix basés sur des conditions différentes. Par exemple, supposez que vous ayez à calculer, pour une valeur de x, la valeur de la fonction f définie par :

$f(x) = x^2 + 1 \quad$ si $x \leqslant 0$
$f(x) = 2x + 1 \quad$ si $0 < x < 2$
$f(x) = x^2 + x - 1$ si $x \geqslant 2$

Bien qu'il s'agisse là d'un choix entre trois possibilités, il n'est pas possible d'utiliser directement un ON GO TO. Pour cela, il nous faut d'abord préparer cette instruction, par exemple, en donnant à une variable K l'une des valeurs 1, 2 ou 3 en fonction de la valeur de x. Vous pourriez écrire :

```
100     INPUT X
110     IF X > 0 THEN 140
120        LET K = 1
130     GOTO 180
140     IF X > 2 THEN 170
150        LET K = 2
160     GOTO 180
170        LET K = 3
180 REM
190     ON K GOTO 200,220,240
200        LET Y = X^2+1
210     GOTO 250
220        LET Y = 2*X+1
230     GOTO 250
240        LET Y = X^2+X-1
250 REM
260     PRINT "VALEUR DE LA FONCTION =";Y
Ok
```

Peut-être remarquez-vous qu'il était plus facile d'écrire ce programme sans instruction ON GO TO, par exemple comme ceci :

```
100     INPUT X
110     IF X > 0 THEN 140
120        LET Y = X^2+1
130     GOTO 180
140     IF X > 2 THEN 170
150        LET Y = 2*X+1
160     GOTO 180
170        LET Y = X^2+X-1
180 REM
190     PRINT "VALEUR DE LA FONCTION =";Y
Ok
```

D'une manière générale, il vous arrivera très souvent de programmer un choix multiple sans cette instruction ON GO TO.

EXERCICES

1. A quoi sert l'instruction ON GO TO ?

2. VRAI ou FAUX ?

a) Après ON, on ne peut trouver que le nom d'une variable.

b) Après ON, on peut trouver n'importe quelle expression arithmétique Basic.

c) Après GO TO, on trouve un seul numéro de ligne.

d) Après GO TO, on trouve toujours trois numéros de ligne.

e) Après GO TO, on trouve autant de numéros de ligne qu'on le souhaite.

f) Dans les numéros de ligne suivant les mots GO TO, il ne peut pas y en avoir deux identiques.

3. Que font ces instructions :

a) 50 ON X GO TO 100, 300, 200, 140

b) 70 ON K + 5 GO TO 300, 350, 450

4. Que fait l'instruction :

```
100 ON X GO TO 150, 200, 250, 300
```

a) si X vaut 1

b) si X vaut 1,9

c) si X vaut 4,2

5. Que fait cette instruction :

```
100 ON Y ^2 - 1 GO TO 200, 300, 400
```

a) si Y vaut 2

b) si Y vaut 1,5

6. Quelles sont les valeurs possibles de X pour que l'instruction suivante fonctionne correctement :

```
80 ON X GO TO 100, 150, 200, 400
```

7. Même question avec :

```
60 ON 2 * X + 1 GO TO 100, 300, 400
```

8. Que risque-t-il de se passer si X vaut 1 ou 3, si l'on exécute :

```
100 ON X ^2 - 2 GO TO 130, 160
```

9. Par quoi pouvez-vous remplacer :

```
100 ON K GO TO 200, 200, 200
```

XIII
QUELQUES FONCTIONS NUMÉRIQUES BASIC

1. INTRODUCTION

Nous avons appris à écrire des expressions arithmétiques Basic contenant les opérations : addition, soustraction, multiplication, division et élévation à la puissance. En fait, en mathématiques, il vous arrive fréquemment d'utiliser également des «fonctions». Par exemple, vous vous servez de la fonction :

valeur absolue dans $|ax + b|$
racine carrée dans $\sqrt{x^2 + 1}$
cosinus dans $\cos(\omega t + \varphi)$

Chaque fonction mathématique est désignée par un ou plusieurs symboles, définis de manière universelle. De même, en Basic, vous pourrez utiliser un certain nombre de «fonctions», connues du système. Ainsi, les trois expressions précédentes s'écriront :

```
ABS (A * X + B)   ABS désignant la fonction valeur absolue
SQR (X ^2 + 1)    SQR désignant la fonction racine carrée
COS (W * T + P)   COS désignant la fonction cosinus.
```

L'écriture d'une fonction mathématique pouvait prendre plusieurs aspects :

– un seul symbole

– plusieurs symboles

– un nom particulier suivi ou non de parenthèses :

cos t $\quad$ cos (ω t + φ)

Par contre, nous avons vu que, dans les exemples Basic correspondants, une fonction s'écrit sous la forme d'un nom suivi, entre parenthèses, d'une expression arithmétique portant le nom d'argument. Il en sera toujours ainsi.

2. LA FONCTION ABS

Comme nous l'avons vu dans nos exemples d'introduction, elle correspond à la valeur absolue. Exécutons quelques instructions en mode direct :

```
LET X = – 5.23
LET A = ABS (X)
PRINT X ; A
– 5.23  5.23
Ok
LET X = 5
LET Y = 10
PRINT ABS (X – Y) ; ABS (Y – X)
5   5
Ok
```

3. LA FONCTION SQR

3.1. Rôle

Elle permet le calcul de racine carrée (son nom provient de l'abréviation de SQuare Root). Voici quelques exemples en mode direct :

```
LET X = 4
PRINT SQR (X)
2
Ok
```

```
LET X = 0
PRINT SQR (X)
0
Ok
```

3.2. Attention aux valeurs négatives

Exécutons ce programme :

```
10 LET X = – 1
20 PRINT SQR (X)
RUN
Illegal function call in 20
```

En effet, l'argument de cette fonction (c'est-à-dire l'expression arithmétique sur laquelle elle porte) doit être positif ou nul. Ceci justifie le message obtenu qui se traduit par :

```
Appel illégal de fonction en 20
```

3.3. Exemple d'utilisation

Nous allons réaliser un programme calculant les racines d'une équation du second degré :

$ax^2 + bx + c = 0$

connaissant les valeurs de a, b et c

Nous vous rappelons que, pour cela, il faut :

1. Calculer le discriminant $\Delta = b^2 - 4\,ac$
2. Si $\Delta < 0$ dire qu'il n'y a pas de racines (réelles)
3. Par contre si $\Delta \geqslant 0$ calculer les deux racines par :

$$x_1 = \frac{-b - \sqrt{\Delta}}{2\,a} \qquad x_2 = \frac{-b + \sqrt{\Delta}}{2\,a}$$

Un programme correspondant pourrait être :

```
100 REM     ***** RESOLUTION D'EQUATIONS DU SECOND DEGRE *****
110 REM
120 REM -------- ENTREE DES COEFFICIENTS
130     PRINT "ENTRER LES 3 COEFFICIENTS"
140     INPUT A,B,C
150 REM
160 REM -------- CALCUL DU DISCRIMINANT D
170     LET D = B^2 - 4*A*C
180 REM
190 REM -------- Y A T'IL DES RACINES REELLES?
200     IF D < 0 THEN 260
210 REM     ----- OUI
220         LET X1 = (-B-SQR(D))/(2*A)
230         LET X2 = (-B+SQR(D))/(2*A)
240         PRINT "RACINES =";X1;X2
250     GOTO 280
260 REM     ----- NON
270         PRINT "PAS DE RACINES REELLES"
280 REM
Ok
```

Exécutons le quelques fois :

```
RUN
ENTRER LES 3 COEFFICIENTS
? 1,-1,1
PAS DE RACINES REELLES
Ok
RUN
ENTRER LES 3 COEFFICIENTS
? 1,-3,2
RACINES = 1  2
Ok
```

Il semble que ce programme fonctionne normalement. Cependant, il ne faut pas conclure trop vite, de même qu'en mathématiques, on ne considère pas qu'un théorème est vrai sous prétexte qu'on l'a vérifié pour quelques cas particuliers.

Ici, en examinant d'un peu plus près notre méthode de calcul des racines, nous découvrons que nous n'avons pas tenu compte de la possibilité que a soit nul. En effet, dans ce cas, les expressions (mathématiques) de x_1 et x_2 ne sont pas définies. En toute rigueur, il faudrait, dans un tel cas, résoudre simplement l'équation : $bx + c = 0$. Ceci n'est pas du tout réalisé dans notre programme. A titre d'expé-

rience, voyons donc comment ce dernier va se comporter si on l'exécute avec une valeur nulle pour A :

```
RUN
ENTRER LES 3 COEFFICIENTS
? 0,1,1
Division by zero
Division by zero
RACINES =-1.70141E+38 -1.70141E+38
Ok
```

Les messages obtenus signifiant :

Division par zéro

Malgré tout, l'exécution se poursuit et les résultats obtenus sont faux. Nous vous donnerons quelques explications sur ce point dans le chapitre 19.

Ici, il est facile d'éviter cet ennui, en refusant une valeur nulle pour A et en reposant la question «ENTREZ LES TROIS COEFFICIENTS» jusqu'à ce que la réponse soit satisfaisante.

4. LA FONCTION INT

4.1. Rôle

INT (de l'anglais INTeger qui signifie entier) permet de calculer la partie entière d'une quantité quelconque. Nous vous rappelons que la partie entière d'un nombre est le plus grand entier inférieur à ce nombre. Voyons quelques exemples en mode direct :

```
PRINT INT (4.3) ; INT (4.6)
4  4
```

Notez bien que INT (4.6) est 4 et non 5. De même :

```
PRINT INT (– 4.6) ; INT (– 4.2)
– 5  – 5
```

Ainsi, pour des X négatifs, les expressions :

ABS (INT (X)) et INT (ABS (X))

ne sont pas égales. Ainsi :

```
PRINT ABS (INT (– 1.3)) ; INT (ABS (– 1.3))
2  1
```

4.2. Exemples d'utilisation

4.2.1. Pour arrondir

Si vous souhaitez arrondir un nombre quelconque à l'entier le plus proche, vous ne pouvez pas appliquer directement la fonction INT. En effet, si INT (4.2) vaut bien 4, INT (4.8) vaut également 4, alors que, dans ce dernier cas, nous souhaiterions obtenir 5.

Un moyen simple d'obtenir le résultat escompté consiste à ajouter 0,5 au nombre considéré avant d'en prendre la partie entière. Ainsi INT (4.2 + 0.5) vaut toujours 4 tandis que INT (4.8 + 0.5) vaut 5.

Voici un petit programme réalisant l'arrondi d'un nombre quelconque entré au clavier :

```
100 REM    ARRONDI A L'ENTIER LE PLUS PROCHE
110     INPUT X
120     LET Y = INT(X+.5)
130     PRINT Y
Ok
RUN
? 5.2
 5
Ok
RUN
? 5.95
 6
Ok
RUN
? -1.25
-1
Ok
RUN
? -1.9
-2
Ok
```

4.2.2. Pour arrondir à deux décimales

S'il vous arrive d'effectuer des calculs dont les résultats correspondent à des sommes d'argent exprimées en francs, il peut être

nécessaire de les arrondir au centime le plus proche. Ceci revient à dire que vous devez arrondir un nombre à deux décimales.

Le problème est voisin du précédent. En effet, dire que X est arrondi à deux décimales revient à dire que 100 * X est arrondi à l'entier le plus proche. D'où la méthode : on multiplie le nombre considéré par 100. On ajoute 0,5 et on prend la partie entière du résultat. Enfin, on divise le tout par 100.

Un petit programme d'arrondi à deux décimales d'un nombre quelconque serait :

```
100 REM        ARRONDI D'UN NOMBRE A DEUX DECIMALES
110     INPUT X
120     LET Y = INT(X*100+.5)/100
130     PRINT Y
Ok
RUN
? 12.35678
 12.36
Ok
RUN
? 34.14895
 34.15
Ok
```

4.2.3. Et pour arrondir à p décimales

Si vous devez arrondir un nombre à p décimales, p étant un entier positif ou nul, vous pouvez généraliser la méthode précédente. Il suffit, en effet, de remarquer que le facteur multiplicatif 100 de l'exemple précédent correspond en fait à 10^2. Pour p décimales, il faut simplement utiliser un facteur 10^p. D'où le programme :

```
100 REM        ARRONDI A P DECIMALES
110     PRINT "NOMBRE A ARRONDIR ET NBRE DE DECIMALES"
120     INPUT X,P
130     LET F = 10^P
140     LET Y = INT(X*F+.5)/F
150     PRINT Y
Ok
RUN
NOMBRE A ARRONDIR ET NBRE DE DECIMALES
? 23.5687,1
 23.6
Ok
```

```
RUN
NOMBRE A ARRONDIR ET NBRE DE DECIMALES
? 12.378524,4
 12.3785
Ok
```

Vous notez que ce programme fonctionne également pour des valeurs négatives de p. Par exemple :

```
RUN
NOMBRE A ARRONDIR ET NBRE DE DECIMALES
? 4258.12,-2
 4300
Ok
```

Ici, on obtient l'arrondi à la centaine la plus proche.

4.2.4. Pour savoir si un nombre est entier

Il peut vous arriver d'avoir à vérifier qu'un nombre est bien entier. Ce serait par exemple le cas, dans un programme déterminant si un nombre entré au clavier est premier ou non.

Comment procéder ? Il vous suffit de comparer le nombre avec sa partie entière. S'il y a égalité, c'est que le nombre entré est effectivement entier. Voici un petit exemple de démonstration :

```
100 PRINT "DONNEZ UN NOMBRE"
110 INPUT N
120 IF N = INT(N) THEN 150
130     PRINT N;"N'EST PAS ENTIER"
140     GOTO 170
150 REM
160     PRINT N;"EST ENTIER"
170 REM
Ok
RUN
DONNEZ UN NOMBRE
? 5
 5 EST ENTIER
Ok
RUN
DONNEZ UN NOMBRE
? 5.0012
 5.0012 N'EST PAS ENTIER
Ok
```

Les trois fonctions mathématiques sin, cos et tg sont représentées en Basic par les symboles SIN, COS et TAN.

5.1. Degrés ou radians ?

L'argument de ces trois fonctions est exprimé en RADIANS. Ainsi :

```
PRINT SIN (5)
```

fournit :

```
– 0.958924
```

qui est la valeur du sinus de cinq radians.

Si vous souhaitez obtenir le sinus de cinq degrés, il vous faut exprimer cet angle en radians. Pour cela, il vous suffit de savoir que :

180 degrés = π radians

d'où 1 degré = $\pi/180$ radians

ou réciproquement :

1 radian = $180/\pi$ degrés

Vous voyez que vous avez besoin de la valeur de π. Généralement, celle-ci n'est pas connue de Basic, et vous devez l'introduire vous-même dans votre programme. Pour éviter d'avoir à l'écrire plusieurs fois, vous pouvez affecter cette valeur à une variable, nommée par exemple P1. Il vous suffit alors d'utiliser simplement P1, autant de fois que nécessaire.

Modifions notre dernier exemple, pour calculer le sinus de cinq degrés :

```
LET P1 = 3.141592
LET X = 5 * P1/180
PRINT SIN (X)
0.0871558
Ok
```

5.2. Pythagore s'est-il trompé ?

Théoriquement, quelle que soit la valeur de x, on a toujours :

$$\cos^2 x + \sin^2 x = 1$$

Examinons ceci en Basic en calculant la valeur de l'expression :

$$\cos^2 x + \sin^2 x - 1$$

pour un certain nombre de valeurs réparties régulièrement, par exemple entre 0 et π.

```
100 REM     PYTHAGORE A T-IL TOUJOURS RAISON ?
110 REM
120    LET PI = 3.141592
130    PRINT "COMBIEN DE VALEURS VOULEZ VOUS"
140    INPUT N
150    LET I = PI/(N-1)
160    LET X = 0
170    FOR K = 1 TO N
180       LET E = 1 - SIN(X)^2 - COS(X)^2
190       PRINT X,E
200       LET X = X + I
210    NEXT K
Ok
```

Vous voyez que ce programme détermine d'abord quel est l'écart entre deux valeurs successives de x. Or, s'il y a n valeurs dans l'intervalle $[0, \pi]$, il y a $n - 1$ intervalles, donc deux valeurs sont séparées de $\pi/(n - 1)$.

La variable X est destinée à décrire les n valeurs qui nous intéressent. Au départ, elle est initialisée à 0. Sa valeur est incrémentée de $\pi/(n - 1)$ à chaque tour de la boucle de compteur K.

Exécutons ce programme :

```
RUN
COMBIEN DE VALEURS VOULEZ VOUS
? 9
 0             -1.19209E-07
 .392699       -2.38419E-07
 .785398       -1.19209E-07
 1.1781        -2.23517E-07
 1.5708        -1.19209E-07
 1.9635        -2.23517E-07
 2.35619       -1.19209E-07
 2.74889        5.96046E-08
 3.14159       -1.19209E-07
Ok
```

Vous êtes peut-être surpris de constater que notre expression ne vaut pas exactement 0. En fait, vous êtes ici en présence des notions importantes de représentation approchée des nombres et d'erreur de calcul. Nous en avons déjà parlé dans le chapitre 8, et nous vous donnerons d'autres détails dans le chapitre 18. Pour l'instant, il vous suffit de savoir que :

– Les nombres ne peuvent être représentés en machine qu'avec une certaine précision. Il en va de même, dans la vie courante, où vous vous limitez à un certain nombre de décimales.

– Les calculs, y compris ceux des fonctions, entraînent des erreurs supplémentaires.

5.3. Attention à la tangente

Mathématiquement, tgx n'est «pas définie» quand x vaut $\pm \pi/2$; plus précisément tgx «tend vers l'infini» quand «x tend vers $\pm \pi/2$», ce qui signifie que cette quantité devient de plus en plus grande lorsque x se rapproche de $\pm \pi/2$.

Ce phénomène va se retrouver en Basic. Si vous cherchez à calculer TAN (X) pour X très proche de $\pm \pi/2$, vous obtiendrez un message d'erreur du type :

Division By zéro

Ceci s'explique par le fait que, généralement, TAN (X) est calculé en divisant SIN (X) par COS (X). Cependant, vous pouvez obtenir un autre message si le calcul en est effectué par une autre méthode.

6. FONCTIONS EXPONENTIELLE ET LOGARITHME

Elles sont représentées par les symboles EXP et LOG.

6.1. Pas d'argument trop grand pour l'exponentielle

Écrivons un programme calculant EXP (X) pour :

X = 1, 2, 4, 8, 16, 32, etc...

```
100 LET X = 1
110 PRINT X,EXP(X)
120 LET X = 2*X
130 GOTO 110
Ok
```

Nous avons là une boucle, en apparence infinie. Exécutons-la quand même quelque temps :

```
RUN
 1              2.71828
 2              7.38906
 4              54.5982
 8              2980.96
 16             8.88611E+06
 32             7.89629E+13
 64             6.23514E+27
 128           Overflow
 1.70141E+38
 256           Overflow
 1.70141E+38
 512           Overflow
 1.70141E+38
 1024          Overflow
```

Que s'est-il passé. Pour le comprendre, il vous faut savoir que Basic ne peut conserver en mémoire que des nombres dont la valeur (absolue) ne dépasse pas une certaine limite (très souvent de l'ordre de 10^{38}). Dans ces conditions, dès que EXP (X) dépasse cette valeur, Basic vous le signale par un message qui signifie :

Dépassement de capacité

Très souvent, le résultat est alors remplacé par cette valeur maximum et l'exécution est poursuivie. C'est le cas dans notre exemple, et cela explique que l'on obtienne ensuite toujours la même valeur de EXP (X) alors que X continue d'évoluer.

6.2. Attention aux arguments négatifs dans LOG

Vous savez que la fonction Logarithme n'est définie que pour des valeurs positives. Bien sûr, Basic vérifie que vous respectez ces conditions. C'est ainsi que :

```
PRINT LOG (– 3)
```

vous fournit le message :

```
Illegal function call
```

qui signifie :

```
Appel illegal de fonction
```

REMARQUE :

LOG correspond à la fonction Log (logarithmes népériens). Si vous souhaitez utiliser d'autres logarithmes, il vous est nécessaire d'effectuer des conversions.

7. FONCTION RND ET INSTRUCTION RANDOMIZE

Nous arrivons maintenant à une particularité bien séduisante de Basic qui consiste à «tirer des nombres au hasard». Cela vous permettra, par exemple, de réaliser des programmes de jeux où ce hasard intervient : dés, cartes, master-mind, etc...

7.1. La fonction RND

Son rôle est de fournir une valeur au hasard comprise entre 0 (inclu) et 1 (exclu). En principe, vous pourriez penser qu'une telle fonction ne nécessite pas d'arguments. En fait, peut-être par souci d'avoir une seule façon d'écrire les fonctions, la plupart des basic demandent que RND possède un argument. Sa valeur, dans ce cas, n'a alors aucune importance. (Toutefois, nous vous conseillons de fournir une valeur différente de 0.)

Voyons un premier exemple :

```
100 FOR I = 1 TO 4
110     PRINT RND(1)
120 NEXT I
Ok
RUN
 .245121
 .305003
 .311866
 .515163
Ok
```

Nous obtenons quatre valeurs au hasard, comprises entre 0 et 1. Exécutons à nouveau ce programme :

```
RUN
 .245121
 .305003
 .311866
 .515163
Ok
```

Vous constatez que nous obtenons la même suite de valeurs que précédemment. Ne pensez surtout pas qu'il en irait autrement en modifiant la valeur de l'argument de RND. Ainsi, en choisissant 2 plutôt que 1, nous obtenons exactement les mêmes résultats :

```
100 FOR I = 1 TO 4
110    PRINT RND(2)
120 NEXT I
Ok
RUN
 .245121
 .305003
 .311866
 .515163
Ok
```

Il est cependant possible que vous souhaitiez obtenir une suite différente à chaque exécution. Ce sera le rôle de l'instruction RANDOMIZE.

7.2. Instruction RANDOMIZE

Elle s'écrit :

RANDOMIZE (exp)

où exp désigne une expression arithmétique quelconque. A chaque valeur de exp, correspond une succession différente de valeurs produites par la fonction RND. Examinez cet exemple :

```
100 PRINT "DONNEZ UN NOMBRE QUELCONQUE ENTRE -32000 ET 32000"
110 INPUT N
120 RANDOMIZE(N)
130 FOR I = 1 TO 4
140    PRINT RND(1)
150 NEXT I
Ok
RUN
DONNEZ UN NOMBRE QUELCONQUE ENTRE -32000 ET 32000
? 2
 .0438479
 .891465
 .801263
 .656113
Ok
RUN
DONNEZ UN NOMBRE QUELCONQUE ENTRE -32000 ET 32000
? 5
 .156153
 .819038
 .261596
 .107356
Ok
RUN
DONNEZ UN NOMBRE QUELCONQUE ENTRE -32000 ET 32000
? 2
 .0438479
 .891465
 .801263
 .656113
Ok
```

Vous voyez que RANDOMIZE(5) a permis à RND de produire une suite différente de celle obtenue avec RANDOMIZE(2). Par contre, pour une valeur donnée de N, le programme fournit toujours la même suite de valeurs.

7.3. Remarque importante

Si la fonction RND existe dans tous les Basic, l'écriture de RANDOMIZE peut prendre plusieurs aspects (parfois, on trouve RANDOM). En outre, dans certains cas, l'argument de RND peut être omis, ou au contraire, avoir une signification. Dans ces conditions, nous vous conseillons de consulter le manuel constructeur de l'ordinateur sur lequel vous travaillez.

7.4. Première application : tirage pile ou face

Le problème est de faire en sorte que l'on ait autant de chances de tirer pile que de tirer face. Or la fonction RND nous fournit un nombre compris entre 0 (inclu) et 1 (exclu). Si nous considérons que nous avons obtenu :

pile si la valeur est comprise entre 0 (inclu) et 0,5 (exclu)

face si la valeur est comprise entre 0.5 (inclu) et 1 (exclu)

vous voyez que nous répondons à la question posée.

Voici un programme que exécute un nombre quelconque (entré au clavier) de tirages, en comptant (dans les variables P et F) le nombre obtenu pour chaque cas.

```
100 REM     TIRAGES PILE OU FACE
110 REM
120    PRINT "COMBIEN DE TIRAGES"
130    INPUT N
140    LET F = 0
150    LET P = 0
160    FOR I = 1 TO N
170       IF RND(1) >= .5 THEN 220
180 REM   ------- CAS PILE
190          LET P = P+1
200          PRINT "PILE"
210          GOTO 250
220 REM   ------- CAS FACE
230          LET F = F+1
240          PRINT "FACE"
250 REM
260    NEXT I
270    PRINT "VOUS AVEZ OBTENU";P;"PILES ET";F;"FACES"
Ok
RUN
COMBIEN DE TIRAGES
? 8
PILE
PILE
PILE
FACE
PILE
FACE
PILE
PILE
VOUS AVEZ OBTENU 6 PILES ET 2 FACES
Ok
```

7.5. Seconde application : jeu de dés

Nous devons tirer un nombre au hasard entre 1 et 6. Nous pourrions nous inspirer de l'exemple précédent en divisant l'intervalle 0,1 en 6 (au lieu de 2). Mais, nous pouvons également remarquer que si :

RND(X) est dans l'intervalle [0,1[(0 inclu, 1 exclu)
6 * RND(X) est dans l'intervalle [0,6[(0 inclu, 6 exclu)

et donc

6 * RND(X) + 1 est dans l'intervalle [1,7[(0 inclu, 7 exclu)

Voici un programme qui «simule» plusieurs jets successifs d'un dé et qui compte le nombre de fois où chaque face est sortie. On utilise pour cela un tableau C de dimension 6 dans lequel :

C (1) représente le nombre de fois où le 1 est sorti
C (2) représente le nombre de fois où le 2 est sorti

et ainsi de suite.

```
100     DIM C(6)
110 REM     MISE A ZERO DES COMPTEURS
120     FOR I = 1 TO 6
130        LET C(I) = 0
140     NEXT I
150 REM     ENTREE DU NOMBRE DE JETS
160     PRINT "COMBIEN DE JETS VOULEZ VOUS"
170     INPUT N
180 REM     REALISATION DES N JETS
190     FOR I = 1 TO N
200        LET K = INT(6*RND(1)+1)
210        LET C(K) = C(K) + 1
220     NEXT I
230 REM     ECRITURE DES RESULTATS
240     FOR I = 1 TO 6
250        PRINT C(I);"FOIS LA FACE";I
260     NEXT I
Ok
```

```
RUN
COMBIEN DE JETS VOULEZ VOUS
? 200
 45 FOIS LA FACE 1
 32 FOIS LA FACE 2
 29 FOIS LA FACE 3
 32 FOIS LA FACE 4
 26 FOIS LA FACE 5
 36 FOIS LA FACE 6
Ok
```

EXERCICES

1. Quelle est la forme générale d'écriture des fonctions Basic ?

2. Où peut-on utiliser une fonction Basic ?

3. Ecrire en Basic les expressions :

a) $|ax + b|$

b) $|a| + |b|$

4. Que va-t-il se passer si l'on exécute ce programme :

```
100 LET X = 4
110 FOR I = 1 TO 2
120 PRINT SQR (X)
130 LET X = – X
140 NEXT I
```

5. Ecrire un programme permettant de calculer la longueur h de l'hypoténuse d'un triangle rectangle, à partir des longueurs x et y des deux autres côtés.

6. Qu'obtiendra-t-on si l'on frappe :

a) PRINT INT (8.2)
b) PRINT INT (8.6)
c) PRINT INT (– 7.4)

7. Quels résultats fournira ce programme :

```
100 LET X = 4.6
110 PRINT INT (2 * X)
120 PRINT 2 * INT (X)
```

8. Comment arrondir la valeur contenue dans X à l'entier le plus proche ?

9. Même question pour la dizaine la plus proche.

10. Quels résultats fournira ce programme :

```
100 INPUT X
110 IF X = INT (X) THEN 140
120 PRINT "NON"
130 GO TO 150
140 PRINT "OUI"
150 REM
```

si on lui répond :

a) 4.3

b) 156

11. Écrire un programme qui, à partir de deux valeurs A et B entrées au clavier, calcule de deux manières différentes cos (A + B) et écrit l'écart entre les deux valeurs ainsi obtenues.

On rappelle que :

$$\cos(a + b) = \cos a \cos b - \sin a \sin b$$

12. VRAI ou FAUX ?

a) RND est une fonction Basic

b) RND n'a pas d'arguments

c) l'argument de RND n'a pas de signification

d) RANDOMIZE est une fonction Basic

e) RANDOMIZE doit toujours être suivi d'une expression arithmétique.

13. Comment procéder si l'on souhaite qu'un programme utilisant RND :

a) donne les mêmes résultats à chaque exécution

b) donne des résultats différents à chaque exécution

XIV
BOUCLES CONDITIONNELLES

1. NOTION DE BOUCLE CONDITIONNELLE

Vous avez appris, avec la boucle FOR, à répéter des instructions Basic un nombre donné de fois. C'est ce que nous avons appelé des «boucles avec compteur». En fait, dans de très nombreux problèmes, vous aurez besoin de réaliser une boucle, sans en connaître a priori le nombre de tours.

Considérez ce programme :

```
100      LET S = 0
110 REM ---- BOUCLE JUSQU'A X = 0
120          INPUT X
130          LET S = S + X
140          IF X >< 0 THEN 110
150 REM ---- FIN BOUCLE JUSQU'A
160      PRINT "SOMME DE CES NOMBRES =";S
Ok
```

Il permet d'ajouter à S chaque nombre entré dans X, en s'arrêtant lorsque X est égal à zéro.

Certes, il aurait été possible d'utiliser une boucle FOR en faisant demander, en début de programme, le nombre de valeurs à ajouter. Ceci aurait toutefois été beaucoup moins pratique puisque l'utilisateur aurait dû compter le nombre de valeurs qu'il souhaitait ajouter.

Dans ce chapitre, nous allons apprendre comment programmer de nouveaux types de boucles avec les seules instructions Basic étudiées jusqu'à ici.

2. BOUCLE «JUSQU'A»

2.1. Définition

Elle sert à répéter des instructions jusqu'à ce qu'une certaine condition devienne vraie. C'est ce que nous avons fait dans notre exemple précédent. Son organigramme se présente ainsi :

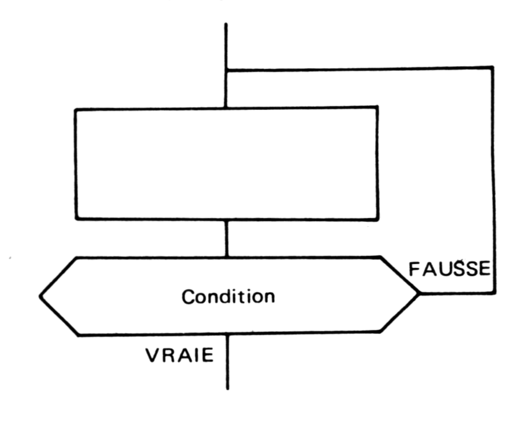


Fig. 14.1.

Le rectangle symbolise ici un nombre quelconque d'instructions.
Vous voyez qu'une telle boucle est toujours exécutée au moins une fois.

2.2. Écriture

Lorsque nous avons introduit la boucle avec compteur, nous avons appris à la réaliser à l'aide d'instructions IF, LET et GO TO, puis nous avons montré comment l'instruction FOR permettait d'écrire les choses plus simplement.

Par contre, en ce qui concerne la boucle jusqu'à, il n'existe pas d'instruction semblable à FOR. Il vous faut la réaliser avec ce

que vous connaissez (comme nous l'avons fait dans l'exemple d'introduction).

2.3. Exemple

Voici un programme qui calcule le capital C obtenu au bout de un an, deux ans, trois ans, . . . lorsque l'on place une certaine somme S à un taux T. Il s'arrête lorsque le capital obtenu dépasse le double de la somme initiale. (La formule donnant C au bout de A années est :

$$C = S(1 + T)^A$$

```
100      PRINT "ENTREZ LA SOMME A PLACER ET LE TAUX"
110      INPUT S,T
120      LET A = 0
130 REM ---- BOUCLE JUSQU'A C > 2*S
140         LET A = A + 1
150         LET C = S*(1+T)^A
160         PRINT "CAPITAL APRES";A;"ANNEES =";C
170         IF C <= 2*S THEN 130
180 REM ---- FIN BOUCLE JUSQU'A
Ok
RUN
ENTREZ LA SOMME A PLACER ET LE TAUX
? 10000,0.12
CAPITAL APRES 1 ANNEES = 11200
CAPITAL APRES 2 ANNEES = 12544
CAPITAL APRES 3 ANNEES = 14049.3
CAPITAL APRES 4 ANNEES = 15735.2
CAPITAL APRES 5 ANNEES = 17623.4
CAPITAL APRES 6 ANNEES = 19738.2
CAPITAL APRES 7 ANNEES = 22106.8
Ok
```

Vous remarquez que nous avons eu besoin d'un compteur A correspondant au nombre d'années. Cependant, le test de fin de boucle ne portant pas sur la valeur de A, nous n'avons pas pu utiliser de boucle FOR.

REMARQUE :

Le capital C obtenu au bout d'un certain nombre d'années peut également s'obtenir en augmentant le capital de l'année précédente des intérêts acquis en un an (c'est-à-dire en multipliant ce capital

par 1 + T). D'où une autre formulation utilisant cette «récurrence» :

```
100      PRINT "ENTREZ LA SOMME A PLACER ET LE TAUX"
110      INPUT S,T
120      LET A = 0
130      LET C = S
140 REM ---- BOUCLE JUSQU'A C > 2*S
150         LET C = C*(1+T)
160         LET A = A + 1
170         PRINT "CAPITAL APRES";A;"ANNEES =";C
180         IF C <= 2*S THEN 140
190 REM ---- FIN BOUCLE JUSQU'A
Ok
```

3. BOUCLE «TANT QUE»

3.1. Définition

Il s'agit, cette fois, de répéter des instructions tant qu'une certaine condition est réalisée. Le test de fin de boucle se trouve alors en début. L'organigramme correspondant se présente ainsi :

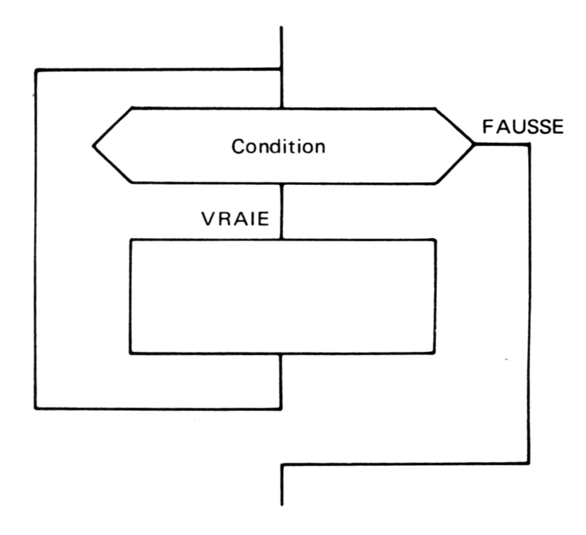


Fig. 14.2

Cette fois, les instructions de la boucle peuvent, éventuellement, n'être exécutées aucune fois.

3.2. Exemple

Le calcul de capital réalisé précédemment peut également s'écrire avec une boucle tant que. En effet, nous pouvons considérer que l'on augmente le capital des intérêts annuels tant que sa valeur ne dépasse pas 2 * S.

```
100      PRINT "ENTREZ LA SOMME A PLACER ET LE TAUX"
110      INPUT S,T
120      LET A = 0
130      LET C = S
140 REM ---- BOUCLE TANT QUE C < 2*S
150         IF C >= 2*S THEN 200
160         LET C = C*(1+T)
170         LET A = A + 1
180         PRINT "CAPITAL APRES";A;"ANNEES =";C
190         GOTO 140
200 REM ---- FIN BOUCLE TANT QUE
Ok
```

4. CONTENU DES BOUCLES CONDITIONNELLES (TANT QUE ET JUSQU'A)

Comme pour les boucles FOR, leur contenu peut être aussi varié et complexe que vous le souhaitez. Notamment, vous pouvez y placer des choix ou d'autres boucles (avec compteur ou non). Qui plus est, lorsque vous programmez de telles boucles conditionnelles, Basic n'y voit que des instructions usuelles. Il ne reconnaît pas une structure de boucle en tant que telle, comme c'était le cas pour l'instruction FOR. Il ne faut donc pas vous attendre à ce qu'il vérifie que certaines règles de logique sont bien respectées. Ainsi, vous pouvez très bien programmer impunément des boucles «jusqu'à» mal imbriquées.

Par ailleurs, vous avez vu qu'il était possible, par maladresse, de réaliser des boucles infinies avec l'instruction FOR. Lorsque vous programmez des boucles conditionnelles, le risque en est encore plus grand. En effet, pour l'éviter, il est nécessaire que le contenu de la boucle puisse, à un moment donné, modifier la condition

servant de test d'arrêt (c'est-à-dire la faire passer de vraie à fausse ou l'inverse).

5. POURQUOI DES STRUCTURES TYPE ?

Depuis le début de ce manuel, nous vous avons proposé un certain nombre de structures types :

– choix (réalisé avec IF et GO TO)

– choix multiple (réalisé avec ON et GO TO)

– boucle avec compteur (réalisée avec FOR et NEXT)

– boucle jusqu'à (réalisée avec IF et GO TO)

– boucle tant que (réalisée avec IF et GO TO)

Pourquoi avons-nous insisté sur celles-ci en particulier, alors qu'il est possible d'en imaginer beaucoup d'autres. C'est qu'en fait, il a été démontré que tout programme pouvait s'écrire à l'aide de deux structures types :

– le choix

– la boucle jusqu'à

Dans ces conditions, il pourrait sembler judicieux de ne pas en utiliser d'autres. Toutefois, il est possible qu'un problème résolu par une boucle jusqu'à s'exprime plus facilement avec une boucle tant que ou avec compteur. Ceci justifie l'existence d'autres structures simples telles que celles évoquées précédemment.

Mais alors, direz-vous, pourquoi ne pas en utiliser d'autres que ces dernières, peut-être mieux adaptées à notre problème ? En réalité, vous découvrirez progressivement que tout programme a besoin :

– de pouvoir être communiqué à autrui (ou même simplement relu par son auteur, quelque temps après qu'il l'ait écrit).

– de pouvoir être corrigé, en cas de comportement anormal.

– d'être modifié pour s'adapter à de nouvelles circonstances.

Vous comprenez alors qu'il soit vivement conseillé d'écrire des programmes aisés à comprendre et à modifier.

Ceci vous sera grandement facilité par l'utilisation d'un nombre limité de structures types. En outre, étant peu nombreuses, il vous sera aisé de rendre chacune d'entre-elles facilement reconnaissable à la première lecture d'un programme. Vous pourrez, par exemple, utiliser des commentaires (remarques) et des décallages d'instructions (comme nous l'avons fréquemment fait dans nos exemples).

6. LA PROGRAMMATION STRUCTURÉE EN BASIC

La programmation dite «structurée» consiste (entre autres choses) à n'utiliser que les structures types évoquées ci-dessus.

La langage «Pascal» possède des instructions permettant de les exprimer simplement, à l'exclusion de toute autre. Ceci n'est plus vrai en Basic, ou rien ne vous empêche de créer n'importe quelle structure de votre choix. En général, d'ailleurs, les programmeurs débutants n'acceptent pas naturellement de se limiter à un nombre restreint de structures. Il semble qu'ils doivent passer par des expériences malheureuses avant de se convertir. Ainsi, certains devront rencontrer, au moins une fois, un programme qu'ils ne parviendront pas à faire fonctionner correctement. D'autres seront alertés par la difficulté qu'ils auront à expliquer à autrui comment «marche» leur programme.

Nous sommes convaincus que vous ne serez pas, du jour au lendemain, des adeptes de la programmation structurée. Aussi, attirons nous votre attention sur ce qui pourra, peut-être, vous protéger des pires difficultés :

– évitez d'avoir plusieurs sorties d'une boucle.

– limitez-vous au maximum dans l'utilisation des GO TO. Lorsqu'ils ne servent pas à réaliser une structure fondamentale, évitez qu'ils ne reviennent «en arrière».

Enfin, les plus convaincus risquent néanmoins d'être rebutés par la difficulté qu'ils auront à exprimer la condition d'arrêt d'une boucle conditionnelle. Très souvent, ils devront, pour cela, utiliser plusieurs instructions IF. Que toutefois, ils se rassurent car nous apprendrons, dans le chapitre 21, comment les «basic étendus» leur donneront des possibilités de simplification.

EXERCICES

1. Qu'appelle-t-on boucle conditionnelle ?

2. Quelle différence y a-t-il entre une boucle avec compteur et une boucle conditionnelle ?

3. A quoi sert la boucle «jusqu'à» ?

4. A quoi sert la boucle «tant que» ?

5. VRAI ou FAUX ?

a) la boucle conditionnelle ne contient jamais de compteur.

b) la boucle conditionnelle contient toujours un compteur.

c) la boucle conditionnelle peut contenir un compteur, mais celui-ci n'est pas utilisé dans le test de fin de boucle.

6. VRAI ou FAUX ? Certaines instructions sont interdites dans une boucle conditionnelle.

7. VRAI ou FAUX ? Il est impossible d'écrire tous les programmes avec un nombre limité de structures types.

8. Quelles structures types avons-nous rencontrées ?

XV
TABLEAUX À DEUX INDICES

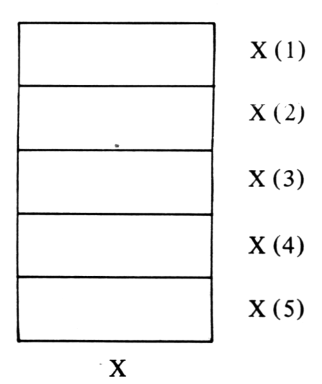
1. INTRODUCTION

Dans le chapitre 11, nous avons introduit la notion de tableau et nous avons dit que l'instruction :

```
DIM X (5)
```

réservait un tableau de cinq éléments, repérés chacun par un indice prenant l'une des valeurs 1, 2, 3, 4 ou 5. Par exemple, ces éléments pouvaient être : X (1), X (2), X (3), X (4) et X (5).

Nous représentions X par un schéma tel :



Mais, nous aurions pu tout aussi bien utiliser celui-ci :

X

X (1) X (2) X (3) X (4) X (5)

De manière analogue, l'instruction :

```
DIM A (3, 2)
```

va réserver un tableau dans lequel chaque élément sera repéré, cette fois, par deux indices :

– le premier pouvant prendre les valeurs 1, 2 ou 3.

– le second pouvant prendre les valeurs 1 ou 2.

Il existera donc 6 (3 x 2) possibilités différentes de choisir les valeurs de ces deux indices. Les six éléments de A pourraient être nommés ainsi :

premier indice	second indice	nom de l'élément
1	1	A (1, 1)
1	2	A (1, 2)
2	1	A (2, 1)
2	2	A (2, 2)
3	1	A (3, 1)
3	2	A (3, 2)

Là encore, nous pouvons illustrer cela par un schéma tel que :

	A (1, 1)	A (1, 2)
A	A (2, 1)	A (2, 2)
	A (3, 1)	A (3, 2)

où l'on représente un tableau formé de trois «lignes» et de deux «colonnes». On pourrait tout aussi bien utiliser ce schéma :

A	A (1, 1)	A (2, 1)	A (3, 1)
	A (1, 2)	A (2, 2)	A (3, 2)

formé de deux lignes et de trois colonnes.

Il est probable que les personnes habituées à la représentation des «matrices» préfèreront le premier schéma. Cependant, dans un cas comme dans l'autre, il ne s'agit que d'une représentation conventionnelle ; la notion de ligne ou de colonne n'est pas directement présente dans la notion de tableau. Tout ce que l'on peut dire, c'est que X possède deux indices, le premier pouvant prendre trois valeurs, le second pouvant en prendre deux. Toutefois, dans la suite, il nous arrivera de parler de lignes et de colonnes d'un tableau. Dans ce cas, nous supposerons que nous nous référons toujours au premier schéma.

2. COMMENT CRÉER ET UTILISER UN TABLEAU A DEUX INDICES

2.1. Réserver la place d'un tableau

Cela se fait, comme pour un tableau à un indice, à l'aide de l'instruction DIM dans laquelle vous précisez, cette fois, les valeurs maximales de chacun des deux indices.

Ainsi :

```
60 DIM C (10, 50)
```

réserve un tableau à deux indices :

– le premier variant de 1 à 10

– le second variant de 1 à 50

C contiendra donc 500 éléments.

Là encore, ces dimensions peuvent être des expressions arithmétiques.

2.2. Introduire des valeurs

Vous pouvez, avec les instructions LET, INPUT ou READ, introduire des valeurs de manière «individuelle», dans chacune des «variables indicées» (éléments) du tableau. Aînsi, ces instructions :

```
100 DIM A (3, 2)
110 LET A (1, 1) = 1
120 LET A (1, 2) = 2
130 LET A (2, 1) = 3
140 LET A (2, 2) = 4
150 LET A (3, 1) = 5
160 LET A (3, 2) = 6
```

ou encore celles-ci :

```
100 DIM A (3, 2)
110 DATA 1, 2, 3, 4, 5, 6
120 READ A (1, 1)
130 READ A (1, 2)
```

```
140 READ A (2, 1)
150 READ A (2, 2)
160 READ A (3, 1)
170 READ A (3, 2)
```

vous permettent de placer les valeurs suivantes dans A :

1	2
3	4
5	6

Mais très souvent, vous serez amenés à utiliser des boucles permettant de répéter une instruction d'affectation (ou READ ou INPUT) pour qu'elle porte successivement sur chacun des éléments du tableau. Ainsi, l'exemple précédent pourrait s'écrire :

```
100 DIM A (3, 2)
110 DATA 1, 2, 3, 4, 5, 6
120 FOR I = 1 TO 3
130       FOR J = 1 TO 2
140             READ A (I, J)
150       NEXT J
160 NEXT I
```

Vous noterez que si l'on avait utilisé ces instructions :

```
110 DIM A (3, 2)
110 DATA 1, 2, 3, 4, 5, 6
120 FOR J = 1 TO 2
130       FOR I = 1 TO 3
140             READ A (I, J)
150       NEXT I
160 NEXT J
```

les mêmes valeurs auraient été introduites dans le tableau, mais dans des éléments différents. Un schéma correspondant serait :

1	4
2	5
3	6

2.3. Utiliser les éléments d'un tableau

Là encore, vous pouvez utiliser les variables indicées dans toutes les expressions arithmétiques. Supposez, par exemple, que nous souhaitions augmenter de 1 la valeur de chaque élément de la première colonne de A, et remplacer chaque élément de la deuxième colonne par son carré. Nous pourrions écrire :

```
200 FOR K = 1 TO 3
210      A (K, 1) = A (K, 1) + 1
220 NEXT K
230 FOR K = 1 TO 3
240      A (K, 2) = A (K, 2) ^2
250 NEXT K
```

3. POUR BIEN PRÉSENTER VOS RÉSULTATS. PRINT TAB

Vous pouvez facilement écrire les éléments du tableau précédent par :

```
100 FOR I = 1 TO 3
110      PRINT A (I, 1), A (I, 2)
120 NEXT I
```

Vous obtenez alors cette présentation :

```
 2              4
 4             16
 6             36
Ok
```

Cependant, il se peut que la «tabulation» qui vous est ainsi imposée ne vous satisfasse pas. Vous avez alors la possibilité de demander que l'écriture se fasse à partir d'une certaine colonne. Par exemple, pour faire écrire chaque variable A (I, 2) à partir de la vingtième colonne, nous écrivons :

```
100 FOR I = 1 TO 3
110      PRINT A (I, 1) ; TAB (20) ; A (I, 2)
120 NEXT I
```

et nous obtenons :

```
 2                    4
 4                   16
 6                   36
Ok
```

Notez que, en toute rigueur, les nombres 4, 16 et 36 sont imprimés à partir de la colonne 21, car Basic place toujours un blanc devant les nombres positifs.

TAB peut être utilisé où vous le souhaitez dans l'ordre PRINT. Dans ces exemples, le numéro de colonne était la constante 20, mais vous pouvez y placer n'importe quelle expression arithmétique. Par exemple, si nous écrivons notre tableau A par :

```
100 FOR I = 1 TO 3
110     PRINT TAB (I) ; A (I, 1) ; TAB (I + 10) ; A (I, 2)
120 NEXT I
```

nous obtenons :

```
 2         4
  4         16
   6         36
Ok
```

Voyons un autre exemple. Soit H un tableau réservé par :

```
DIM H (16, 10)
```

Nous souhaitons l'écrire ainsi :

```
H (1, 1)  H (1, 2)  H (1, 3) . . . . . . . . . .  H (1, 10)
H (2, 1)  H (2, 2)  H (2, 3) . . . . . . . . . .  H (2, 10)
          . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
H (16,1)  H (16,2)  H (16, 3) . . . . . . . . .  H (16, 10)
```

chaque valeur disposant de sept caractères.

Vous pensez peut-être à procéder comme dans l'exemple précédent, en nommant dans une instruction PRINT les dix éléments à écrire sur une même ligne, précédés chacun de leur tabulation. Vous risqueriez d'obtenir une instruction ne tenant pas dans une ligne. En fait, vous pouvez très bien ne nommer dans l'instruction PRINT qu'un seul élément, à condition de faire en sorte que Basic ne passe pas automatiquement à la ligne pour le PRINT suivant. Peut-être vous souvenez-vous qu'il suffit, pour cela, de terminer

votre instruction PRINT par un point-virgule. Bien mais, direz-vous, il faudra changer de ligne de temps en temps (tous les dix éléments). Cela se fait grâce à une instruction PRINT (tout court) qui a pour effet de ne rien écrire, avant de passer à la ligne suivante (car elle n'est pas terminée par point-virgule). D'où les instructions :

```
100 REM *** ECRITURE DU TABLEAU H(16,10) SOUS FORME
110 REM          DE 16 LIGNES DE 10 ELEMENTS ****
120       FOR I = 1 TO 16
130 REM ---- ECRITURE D'UNE LIGNE
140          FOR J = 1 TO 10
150 REM      ---- ECRITURE D'UN ELEMENT
160             PRINT TAB(7*J-6);H(I,J);
170          NEXT J
180 REM      ---- PASSAGE LIGNE SUIVANTE
190          PRINT
200       NEXT I
Ok
```

Vous noterez que TAB (7 * J – 6) permet de se placer sur les colonnes :

1, 8, 15, 22, 29, 36, 43, 50, 57, 64

suivant que J vaut 1, 2, 3, . . ., 10.

4. UN PROBLEME NOUVEAU. CALCUL DE LA SOMME DES ÉLÉMENTS D'UN TABLEAU A DEUX INDICES

4.1. Rappels

Nous avons vu comment calculer la somme des éléments d'un tableau à un indice. Ainsi, pour B de dimension 8, nous écrivions :

```
100 LET S = 0
110 FOR I = 1 TO 8
120       LET S = S + B (I)
130 NEXT I
```

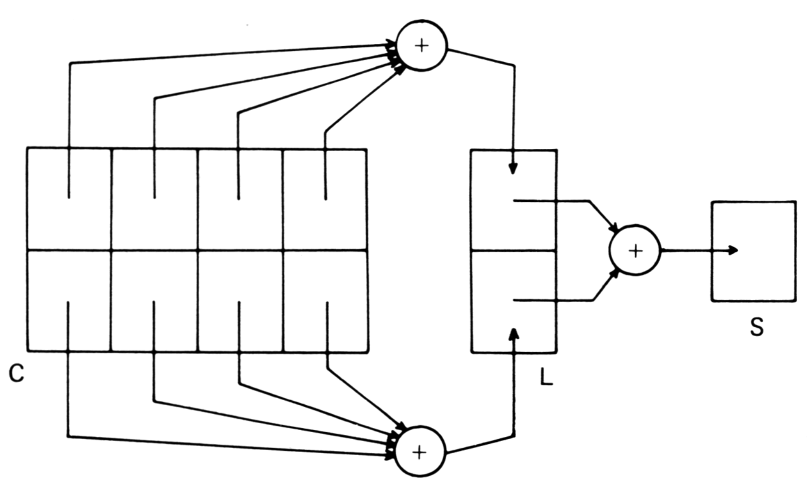
L'instruction 100 «initialise» une variable S à zéro. La boucle FOR permet d'ajouter chacun des éléments B (I) à S. Au bout du compte, nous obtenons dans S le résultat cherché.

4.2. Avec deux indices – première idée.

Généralement, ce problème est perçu comme difficile par les débutants. Souvent, ils pensent que pour calculer la somme des éléments d'un tableau C (2, 4), il leur faudra préalablement calculer la somme de chacune des lignes (ou de chacune des colonnes). Ceci nous obligerait alors à créer un tableau de deux éléments destiné à contenir les deux sommes de chacune des lignes. Par exemple, nous serions amenés à écrire, en supposant qu'un tableau L a été réservé par : DIM L (2)

```
100 REM ---- SOMME DES 2 LIGNES
110      FOR I = 1 TO 2
120 REM       ---- SOMME DE LA LIGNE I DANS L (I)
130           LET L (I) = 0
140           FOR J = 1 TO 4
150                LET L (I) = L (I) + C (I, J)
160           NEXT J
170      NEXT I
180 REM
190 REM ---- SOMME DES ELEMENTS DE L
200      LET S = 0
210      FOR I = 1 TO 2
220           LET S = S + L (I)
230      NEXT I
```

Le déroulement peut être schématisé ainsi :



4.3. Une meilleure solution

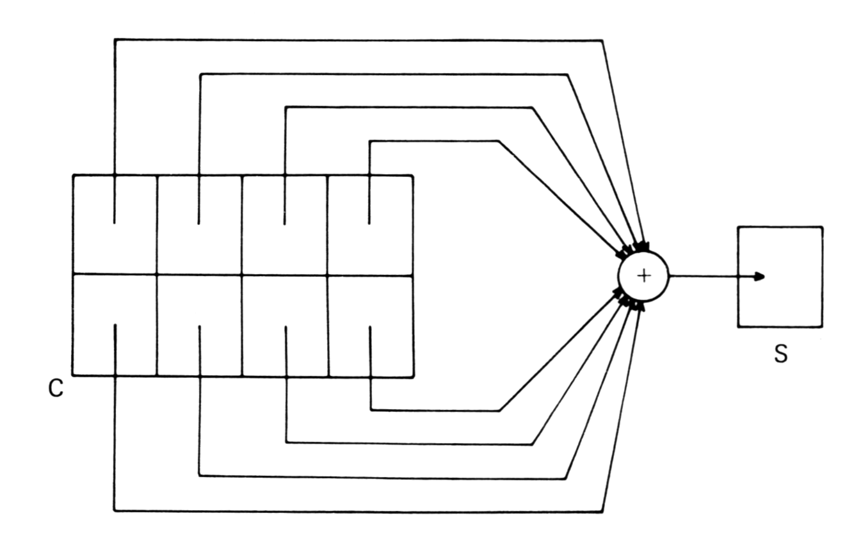
En fait, nous pouvons considérer que l'on doit calculer, dans S, la somme de huit éléments, et éviter de faire des sommes intermédiaires de ligne. Le programme doit donc se présenter comme une initialisation de S à zéro, suivie d'instructions permettant d'ajouter chacune des huit valeurs (deux fois 4) de C à S. D'où les instructions :

```
100 LET S = 0
110 FOR I = 1 TO 2
120       FOR J = 1 TO 4
130             LET S = S + C (I, J)
140       NEXT J
150 NEXT I
```

Vous voyez que l'instruction 130 a été repétée 8 fois, permettant de prendre en compte, successivement :

```
C (1, 1)  C (1, 2)  C (1, 3)  C (1, 4)
C (2, 1)  C (2, 2)  C (2, 3)  C (2, 4)
```

ce qui peut se schématiser ainsi :



EXERCICES

1. Soit un tableau réservé par :

DIM X (4, 3)

a) Combien doit-on utiliser d'indices pour repérer un élément de X ?

b) Quelles valeurs peut prendre le premier indice ?

c) Quelles valeurs peut prendre le second indice ?

d) Combien d'éléments ce tableau contient-il ?

2. Écrire les instructions permettant de placer la valeur 1 dans tous les éléments d'un tableau réservé par :

DIM R (20, 50)

3. Écrire les instructions permettant de placer dans un tableau T les valeurs suivantes :

5	12	8	1	6
6	9	2	15	8

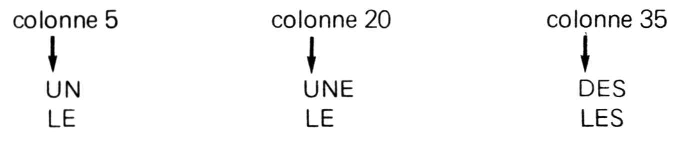
en utilisant une instruction DATA.

4. Écrire les instructions permettant de placer dans un tableau X les chaînes de caractères suivantes :

UN	UNE	DES
LE	LA	LES

en utilisant une instruction DATA.

5. Écrire le contenu du tableau précédent, en présentant les résultats ainsi :



6. Même question en présentant les résultats comme ceci :



7. Écrire les instructions permettant de placer dans un tableau T les valeurs suivantes :

5	10	15	20
25	30	35	40
45	50	55	60

sans utiliser d'instruction DATA.

XVI
LES FONCTIONS UTILISATEUR

1. INTRODUCTION

Dans le chapitre 13, nous avons appris à utiliser un certain nombre de fonctions numériques telles que SIN, EXP, RND, etc... Il est possible que celles-ci ne vous suffisent pas et que vous souhaitiez en créer de nouvelles.

En mathématiques, vous avez l'habitude de définir une fonction en écrivant, par exemple :

$$g(x) = x^3 + x \cos x - 1$$

Vous avez ainsi écrit un «modèle» pour la fonction g en utilisant pour cela une variable nommée x. En fait, cette dernière n'a pas d'importance en elle-même. Vous auriez pu définir la même fonction en écrivant :

$$g(u) = u^3 + u \cos u - 1$$

Nous exprimons souvent cela en disant que x (ou u) est une variable muette.

Cette fonction g ayant été créée, vous pouvez alors l'utiliser dans des calculs en écrivant :

$$g(a) \text{ ou } g(2) \text{ ou } g(t) + g(2t) \text{ etc...}$$

Cette fois, la valeur à laquelle vous appliquez la fonction (a, 2, t ou 2 t dans nos exemples) a une signification effective, et vous ne pouvez plus indifféremment remplacer a par b ou t.

Ces notions vont se retrouver, pratiquement sous la même forme, dans le langage Basic.

2. NOTION DE FONCTION BASIC DÉFINIE PAR L'UTILISATEUR

Comme nous l'avons fait en introduction pour les fonctions mathématiques, nous allons distinguer :

– la définition de la fonction

– son utilisation

2.1. Définition d'une fonction

Elle se fait, en écrivant, par exemple :

```
100 DEF FNA (X) = X ^2 + 1
```

X est, là encore, nommée «variable muette» ou «argument muet». On définirait la même fonction, en écrivant :

```
100 DEF FNA (U) = U ^2 + 1
```

Par contre, vous ne pouvez pas écrire :

```
100 DEF FNA (1) = 2
```

D'une manière générale, nous trouvons, à gauche du signe =, le nom donné à la fonction suivi, entre parenthèses, d'un nom de variable. Le nom de la fonction est formé de FN suivi d'une lettre ou d'une lettre et d'un chiffre. A droite du signe égal, nous trouvons l'expression de la fonction. Elle fait intervenir la variable muette mentionnée à gauche. On peut y faire référence à d'autres fonctions Basic. Ainsi, notre fonction g de l'introduction pourrait être définie par :

```
100 DEF FNG (X) = X ^3 + X * COS (X) – 1
```

REMARQUE IMPORTANTE :

La variable muette apparaissant dans cette définition peut très bien porter le nom d'une variable utilisée par ailleurs dans le programme. Cela n'affectera, en aucun cas, la valeur de cette dernière.

2.2. Utilisation d'une fonction

Elle se fait dans n'importe quelle expression arithmétique en donnant un argument nommé alors «effectif» (par opposition à muet) qui peut être n'importe quelle expression arithmétique Basic. Voici quelques exemples d'utilisation des fonctions FNA et FNG précédentes :

```
200 LET X = FNA (5)
260 LET X = FNA (T) + 2 * X * COS (T)
400 LET Y = FNG (A * X + 3)
450 PRINT FNA (3) + 2 * FNG (A + 1)
```

2.3. Exemples

a) Voici un petit programme réalisant l'arrondi d'un nombre à deux décimales :

```
200 DEF FNA2(X) = INT(X*100+.5)/100
210 PRINT FNA2(11.248);FNA2(1.412)
Ok
RUN
 11.25  1.41
Ok
```

b) Voici un exemple montrant que la variable servant à la définition de la fonction est bien muette :

```
100 DEF FNA(X) = X^3 + 1
110 LET X = 4
120 LET Y = FNA(0)
130 LET Z = FNA(X)
140 PRINT Y;Z
Ok
RUN
 1  65
Ok
```

La variable X apparaissant en 100 est muette. Elle n'a pas d'existence réelle. Aucun conflit n'a lieu avec la variable effective X utilisée plus loin (en 110 et 130). Lorsqu'en 120, Basic doit utiliser la définition de FNA pour calculer FNA (0), il utilise effectivement l'instruction 100, mais en remplaçant la variable muette X par 0. De même en 130, Basic utilise à nouveau la définition de FNA, en remplaçant la variable muette X par l'argument effectif X (qui vaut alors 4).

3. QUAND L'EXPRESSION DÉFINISSANT LA FONCTION FAIT INTERVENIR DES VARIABLES NON MUETTES

Dans les exemples précédents, l'expression définissant la fonction ne faisait intervenir que la variable (muette) mentionnée à gauche du signal égal. Il n'en va plus de même dans :

```
200 DEF FNH (Y) = A * Y + 5
```

Si Y est bien une variable muette, ce n'est pas le cas de A qui désigne, en fait, une variable (ordinaire). Autrement dit, lorsque l'on fera référence, dans le programme, à :

```
FNH (Z)
```

Basic utilisera l'instruction 200, en remplaçant, bien sûr, Y par Z. Par contre, il se servira de la valeur que A possèdera à ce moment là.

Exemple 1

```
100 DEF FNH(Y) = A*Y + 5
110 LET X = 4
120 LET A = 2
130 PRINT FNH(X)
140 LET A = 3
150 PRINT FNH(X)
Ok
RUN
 13
 17
Ok
```

L'exécution de l'instruction 130 a entraîné l'utilisation de la définition de FNH, en remplaçant Y par X. La valeur de A est alors 2. Par contre, en 150, le même mécanisme se déroule, mais cette fois A vaut 3.

Exemple 2 :

Arrondi a p décimales

```
100 DEF FNA(X) = INT(X*10^P+.5)/10^P
110 LET X = 126.762
120 FOR P = -2 TO 3
130     PRINT FNA(X)
140 NEXT P
Ok
RUN
 100
 130
 127
 126.8
 126.76
 126.762
Ok
```

4. PRÉCAUTIONS

L'instruction DEF doit être rencontrée avant son utilisation. Ainsi, ce programme conduit à un message d'erreur signifiant :

Fonction utilisateur non définie en 110

```
100 LET X = 3
110 PRINT FNA(X)
120 DEF FNA(Z) = Z^2+1
Ok
RUN
Undefined user function in 110
Ok
```

Si vous faites une erreur de syntaxe dans la définition d'une fonction, vous ne le découvrirez que lors de sa première utilisation, comme dans cet exemple :

```
100 DEF FNG(I) = 3I^2+1
110 LET X = 2
120 LET Y = FNG(X)
130 PRINT Y
Ok
RUN
Syntax error in 120
Ok
120
```

Qui plus est, Basic vous mentionne la ligne où vous cherchez à utiliser la fonction incorrecte (ici 120), et non pas la ligne de définition (100) qui est pourtant celle où il y a véritablement une erreur (ici, on a omis le symbole * entre 3 et I).

EXERCICES

1. VRAI ou FAUX ? ces instructions sont équivalentes :

```
200 DEF FNH (X) = X ^3 + X ^2 + X
200 DEF FNH (Y) = Y ^3 + Y ^2 + Y
```

2. Même question avec :

```
200 DEF FNP (X) = COS (X) + SIN (X)
200 DEF FNP (0) = 1
```

3. Qu'est-ce qu'une variable muette ?

4. Quels résultats produit ce programme :

```
100 DEF FNA (X) = 2 * X – 1
110 LET Y1 = 0
120 LET Y2 = 1
130 PRINT FNA (Y1) ; FNA (Y2)
```

5. Même question avec :

```
100 DEF FNH (Y) = 2 * Y ^2 + Y
110 LET Y = 2
120 LET F1 = FNH (Y)
130 LET Y = 1
140 LET F2 = FNH (Y)
150 PRINT FNH (0) ; F1 ; F2
```

6. VRAI ou FAUX ? dans le programme précédent :

a) Y est une variable muette.

b) Y est un argument effectif.

7. Dans le programme précédent, quelle différence y-a-t-il entre la variable Y de la ligne 100 et la variable des lignes 110 ou 130.

8. VRAI ou FAUX ? Dans la définition d'une fonction, l'expression apparaissant à droite du signe égal ne peut utiliser que la variable muette mentionnée à gauche.

9. Quels résultats produit ce programme :

```
100 DEF FNC (X) = X ^2 + A * X
110 LET A = 2
120 LET Y = 1
130 PRINT FNC (Y)
140 LET A = 3
150 PRINT FNC (Y)
```

10. Même question avec :

```
100 DEF FNC (X) = X ^3 + A * X
110 LET A = 2
120 PRINT FNC (A)
130 LET X = 2
140 PRINT FNC (X)
```

11. Quelle erreur comporte ce programme ?

```
100 LET X = 2
110 LET Y = FNC (X) + COS (X)
120 PRINT Y
130 DEF FNC (A) = SIN (A) + COS (A/2)
```

Quel message obtiendra-t-on si on l'exécute ?

12. Mêmes questions avec :

```
100 DEF FNL (Z) = 2 EXP (Z/2) + LOG (Z + 1)
110 LET X = 5
120 PRINT FNL (X)
```

XVII
SOUS-PROGRAMMES

1. NOTION DE SOUS-PROGRAMME

Supposez que vous ayez besoin, en différents endroits d'un programme, de faire entrer une valeur au clavier. Si, en outre, vous souhaitez vérifier qu'elle est entière (en en demandant une nouvelle dans le cas contraire, et ceci jusqu'à ce que la réponse soit correcte), vous pourriez vous trouver dans la situation suivante :

```
-----
200 PRINT "DONNEZ UNE VALEUR ENTIERE"
210 INPUT N
220 IF N<>INT (N) THEN 200
-----
-----
-----
360 PRINT "DONNEZ UNE VALEUR ENTIERE"
370 INPUT N
380 IF N<> INT (N) THEN 360
-----
```

Les instructions 360 à 380 sont les mêmes que les instructions 200 à 220. Vous pouvez trouver cela regrettable et espérer ne pouvoir les écrire qu'une seule fois. Basic vous offre une solution prati-

que à ce problème dans la notion de sous-programme. C'est ainsi que notre exemple pourrait s'écrire :

```
200 GOSUB 1000
210 -----
    -----
360 GOSUB 1000
370 -----
    -----
1000 REM SOUS-PROGRAMME D'ENTRÉE D'UNE VALEUR ENTIERE
1010 PRINT "DONNEZ UNE VALEUR ENTIERE"
1020 INPUT N
1030 IF N <> INT (N) THEN 1010
1040 RETURN
```

Que se passe-t-il ? L'instruction 200 demande à Basic d'aller exécuter le sous-programme situé en 1000 (GOSUB étant l'abréviation de GO to SUBroutine, c'est-à-dire aller au sous-programme). Vous pourriez penser que son rôle est le même que celui d'une instruction GO TO 1000. En fait, GOSUB 1000 réalise effectivement un «branchement en 1000», mais en même temps, Basic conserve le numéro de l'instruction suivante (ici 210). Les instructions 1000 à 1030 sont alors exécutées, éventuellement répétées jusqu'à ce que la réponse soit correcte. En 1040, Basic rencontre une instruction RETURN (c'est-à-dire retour) lui demandant de quitter le sous-programme. Il revient alors à l'instruction 210 (dont il avait conservé le numéro lors de l'exécution de GOSUB).

En 360, nous retrouvons exactement le même mécanisme, avec cette différence que c'est le numéro 370 (au lieu de 210) qui sera conservé par Basic lors de l'exécution de GOSUB.

REMARQUE :

On dit souvent que GOSUB est une instruction «d'appel de sous-programme».

2. QUELQUES REGLES ET PRÉCAUTIONS

2.1. Bien mettre en évidence le début d'un sous-programme

Vous remarquez qu'aucune instruction ne précise à quel endroit commence un sous-programme. Pour y voir clair, nous vous conseillons d'y placer, comme dans notre exemple, une ou plusieurs instructions REM.

De même, il vaut mieux éviter de placer un sous-programme au début ou au milieu d'un programme, comme dans cet exemple :

```
100 LET X = 20
110 REM SOUS-PROGRAMME .....
    -----
    -----
200 RETURN
210 ------
    -----
```

Ici, après avoir exécuté l'instruction 100, Basic va exécuter les instructions 110 à 200 du sous-programme. Cependant, comme il n'y est pas arrivé par un GOSUB, il ne sait pas à quelle instruction retourner lorsqu'il rencontre le RETURN. Ceci se traduira par un message d'erreur du type :

```
RETURN without GOSUB in 200
```

Vous pouvez songer à éviter cela, en plaçant une instruction :

```
105 GO TO 210
```

Toutefois, nous vous conseillons, pour des raisons de clarté, de placer plutôt votre sous-programme à la fin de votre programme.

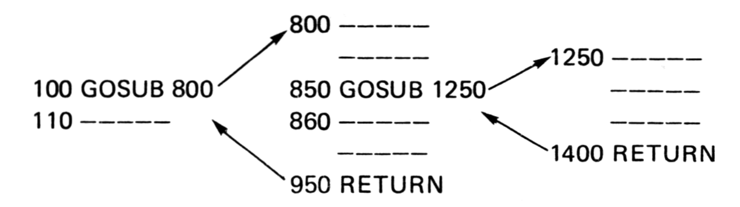
Enfin, comme un sous-programme n'a pas de «début» visible par Basic, il est, en principe, autorisé de l'aborder (par GOSUB) à partir de n'importe quelle instruction, et non seulement à partir de la première. En voici un exemple :

```
-----
GOSUB 800
-----
-----
GOSUB 850
     -----
     -----
800 REM SOUS-PROGRAMME .....
     -----
850 -----
     -----
RETURN
```

Cependant, pour des raisons de simplicité et de compréhension, nous vous conseillons de ne pas abuser de cette possibilité.

2.2. Imbrication de sous-programmes

Un sous-programme peut, à son tour, en appeler un autre. Voyez cet exemple :



L'instruction 100 envoie au sous-programme commençant en 800, et Basic conserve le numéro 110. A la rencontre de l'instruction 850, Basic va alors exécuter le sous-programme situé en 1250, après avoir conservé le numéro 860. L'instruction 1400 lui demande alors de revenir au dernier numéro enregistré, c'est-à-dire ici 860. De même l'instruction 950 le fait revenir en 110.

D'une manière générale, l'instruction RETURN renvoie à l'instruction suivant le dernier GOSUB qui a été exécuté.

Là encore, soyez prudents dans l'utilisation de ces possibilités et songez à bien distinguer les sous-programmes les uns des autres.

3. EXEMPLE D'UTILISATION

Supposez que nous souhaitions écrire un programme entraînant un élève à effectuer des multiplications. Nous avons besoin, pour cela, de «tirer au hasard» deux nombres entiers (par exemple inférieurs à 1000). Nous allons même demander de rejeter les nombres qui sont multiples de 10, ou ceux n'ayant que deux chiffres. Ceci nous conduit à ce sous-programme :

```
500 REM ********** SOUS PROGRAMME **********
510 REM ** TIRAGE D'UN NOMBRE DE 3 CHIFFRES **
520 REM ************************************
530 REM ---- BOUCLE JUSQU'A X > 100
540 REM          ET NON MULTIPLE DE 10
550          LET X = INT (999*RND(1)+1)
560          IF INT(X/10) = X/10 THEN 550
570          IF X < 100          THEN 550
580 REM ---- FIN BOUCLE JUSQU'A
590          RETURN
600 REM ******* FIN SOUS PROGRAMME *********
Ok
```

Nous pouvons alors utiliser ce sous-programme pour réaliser le choix de deux nombres au hasard. Toutefois, à chaque appel, nous obtiendrons une valeur dans X. Or, ici, nous avons besoin que chaque résultat apparaisse dans une variable différente. Pour cela, il est nécessaire, après le retour du sous-programme, de recopier le contenu de X dans une variable de notre choix.

Voici un programme qui propose des multiplications à l'élève jusqu'à ce qu'il souhaite s'arrêter. En outre, il compte les bonnes et les mauvaises réponses et il en fournit le total à la fin.

```
100 REM **** ENTRAINEMENT A LA MULTIPLICATION ****
110 REM
120 REM ---- INITIALISATION DES COMPTEURS
130      LET N1 = 0
140      LET N2 = 0
150 REM ---- BOUCLE JUSQU'A DEMANDE D'ARRET
160 REM        ---- CHOIX DE NOMBRES P1 ET P2
170           GOSUB 500
180           LET P1 = X
190           GOSUB 500
200           LET P2 = X
210           LET P = P1*P2
220 REM        ---- INTERROGATION DE L'ELEVE
230           PRINT "COMBIEN FONT";P1;"X";P2;"?"
240           INPUT R
250 REM        ---- ANALYSE DE LA REPONSE
260           IF R = P THEN 330
270 REM           ---- REPONSE FAUSSE
280              PRINT "TU T'ES TROMPE"
290              PRINT "LA BONNE REPONSE EST";P
300              LET N2 = N2 + 1
310             GOTO 350
320 REM        ---- REPONSE EXACTE
330              PRINT "TA REPONSE EST EXACTE"
340              LET N1 = N1 + 1
350 REM        ---- DEMANDE D'ARRET
360           PRINT "VEUX TU CONTINUER ? (OUI OU NON)"
370           INPUT R$
380           IF R$ = "OUI" THEN 170
390 REM ---- FIN BOUCLE JUSQU'A
400 REM ---- ARRET APRES ECRITURE DES COMTEURS
410      PRINT "TU AS OBTENU"
420      PRINT N1;"BONNES REPONSES"
430      PRINT N2;"MAUVAISES REPONSES"
440      END
500 REM *********** SOUS PROGRAMME ***********
510 REM ** TIRAGE D'UN NOMBRE DE 3 CHIFFRES **
520 REM ************************************
530 REM ---- BOUCLE JUSQU'A X > 100
540 REM          ET NON MULTIPLE DE 10
550         LET X = INT (999*RND(1)+1)
560         IF INT(X/10) = X/10 THEN 550
570         IF X < 100          THEN 550
580 REM ---- FIN BOUCLE JUSQU'A
590         RETURN
600 REM ******** FIN SOUS PROGRAMME *********
Ok
```

```
RUN
COMBIEN FONT 245 X 305 ?
? 74725
TA REPONSE EST EXACTE
VEUX TU CONTINUER ? (OUI OU NON)
? OUI
COMBIEN FONT 312 X 515 ?
? 160780
TU T'ES TROMPE
LA BONNE REPONSE EST 160680
VEUX TU CONTINUER ? (OUI OU NON)
? OUI
COMBIEN FONT 789 X 497 ?
? 392133
TA REPONSE EST EXACTE
VEUX TU CONTINUER ? (OUI OU NON)
? OUI
COMBIEN FONT 364 X 984 ?
? 358176
TA REPONSE EST EXACTE
VEUX TU CONTINUER ? (OUI OU NON)
? NON
TU AS OBTENU
 3 BONNES REPONSES
 1 MAUVAISES REPONSES
Ok
```

4. L'INSTRUCTION ON – GOSUB

Nous avons vu que l'instruction :

ON K GO TO 100, 200, 250

demandait à Basic de poursuivre l'exécution en 100, 200 ou 250 suivant la valeur de K.

De la même manière, l'instruction :

ON K GOSUB 1000, 1100, 2000

permet d'appeler l'un des sous-programmes placés en 1000, 1100 ou 2000 suivant la valeur de K. Comme il le faisait pour GOSUB, Basic conserve le numéro de l'instruction suivant le ON GOSUB afin d'y revenir lorsqu'il rencontrera l'instruction RETURN du sous-programme correspondant.

EXERCICES

1. Qu'est-ce qu'un sous-programme ?

2. A quoi sert l'instruction GOSUB ?

3. A quoi sert l'instruction RETURN ?

4. VRAI ou FAUX ? on peut entrer dans un sous-programme par une instruction GO TO.

5. Quelle erreur a-t-on commise dans ce programme :

```
100 REM SOUS-PROGRAMME D'ENTREE D'UN NOMBRE
110 PRINT "DONNEZ UN NOMBRE ENTIER"
120 IF N <> INT (N) THEN 110
130 RETURN
140 GOSUB 100
150 END
```

6. VRAI ou FAUX ? un sous-programme peut en appeler un autre.

7. A quoi sert l'instruction ON – GOSUB ?

8. Quelle différence y-a-t-il entre ON – GOTO et ON – GOSUB ?

AVERTISSEMENT
concernant les chapitres 18 à 21

Dans les précédents chapitres, nous avons étudié le Basic correspondant à une norme précise, datant de 1978 et portant la référence : ANSI-X3-60 1978. Celle-ci décrit un «basic minimum» qui doit être utilisable sur toute machine travaillant dans ce langage (nous l'avons souvent appelé également «basic standard».

En pratique, tous les micro-ordinateurs proposent davantage de possibilités que n'en prévoit cette norme. Comme la plupart des fabricants ont fait appel à la même firme (Microsoft) pour écrire les programmes interprêteurs Basic, les langages proposés sont très proches les uns des autres. Ainsi, il existe presque une standardisation de fait.

Dans les chapitres 18 à 21, nous décrivons les extensions par rapport à la norme ANSI dont nous savons qu'elles existent pratiquement sur tous les micro-ordinateurs. Dans certains cas, nous avons été amenés à préciser des notions déjà évoquées afin de vous aider à mieux apprécier les nouvelles possibilités du Basic que nous qualifierons d'étendu. (Vous l'entendrez souvent dénommé «basic microsoft).

XVIII
COMPLÉMENTS SUR LES VARIABLES

1. NOTION DE CODAGE

1.1. Toute information est codée

Dans la vie courante, lorsque vous échangez des informations, vous utilisez, la plupart du temps inconsciemment, un «codage». Ainsi, pour transmettre à quelqu'un le nombre de personnes que vous avez comptées dans une pièce, vous pouvez :

– *l'écrire en chiffres* : 13. Vous utilisez pour cela deux chiffres, chacun étant choisi parmi les 10 symboles 0, 1, 2, ..., 9.

– *l'écrire en lettres* : TREIZE. Vous utilisez alors six lettres, chacune étant choisie parmi les 26 lettres de l'alphabet.

D'une manière générale, tout échange d'information nécessite un codage. Celui-ci est purement conventionnel. Ainsi, avec les mêmes lettres, il est possible de coder la même information de plusieurs manières différentes. Il vous suffit de songer aux diverses langues étrangères qui emploient le même alphabet.

Deux éléments interviennent dans un codage :

– le nombre de positions (nombre de chiffres, de lettres, ...) utilisées pour exprimer l'information.

– le nombre de valeurs différentes que peut prendre chaque position. On l'appelle «moment». Ainsi, dans nos exemples, 13 était écrit

avec un code à 10 moments, alors que TREIZE utilisait un code à 26 moments.

Considérons notre manière d'écrire les nombres dans le système décimal (code à 10 moments) :

Sur une position, nous pouvons écrire 10 nombres différents (les nombres de 0 à 9).

Sur deux positions, nous pouvons écrire $10 \times 10 = 10^2 = 100$ nombres différents (les nombres de 00 à 99).

Sur trois positions, nous pouvons écrire $10^3 = 1000$ nombres différents,

etc...

Avec un code à 26 moments, nous pouvons coder :

– sur une position : 26 informations différentes

– sur deux positions : 26^2 informations différentes

– sur trois positions : 26^3 informations différentes, etc...

1.2. Le bit

Dans le cas des ordinateurs, il se trouve que ce sont les codes à deux moments qui sont les plus économiques. Chaque position d'un tel code s'appelle un «BIT». (Ce mot provient de BInary digiT). Il faut donc coder toute information sur un certain nombre de bits.

Ainsi, les nombres ou les chaînes de caractères seront représentés, en mémoire, par une suite de bits.

Vous voyez que :

– sur un bit, vous pouvez coder deux informations différentes. On les note souvent 0 et 1. (Mais ceci est purement conventionnel – on pourrait utiliser n'importe quels symboles comme a et b, ou X et Y ou «oui» et «non»).

– sur 2 bits, vous pouvez en coder $2^2 = 4$

– sur 3 bits, vous pouvez en coder $2^3 = 8$

. . . .

– sur 7 bits, vous pouvez en coder $2^7 = 128$

– sur 8 bits, vous pouvez en coder $2^8 = 256$

1.3. Notion de type

Nous avons dû distinguer deux types de variables :

– les variables numériques

– les variables chaînes de caractères

Chacun de ces deux types correspond en fait à un codage différent. Dans les deux cas, il s'agit bien sûr d'une suite de bits, mais la signification qui lui est attribuée est différente.

Avant d'étudier les différents types existant en Basic étendu, nous allons donner quelques notions sur le codage utilisé pour représenter les deux types du Basic standard.

2. LES TYPES DU BASIC STANDARD

2.1. Le type chaîne de caractères

Chaque caractère est généralement codé sur un octet (c'est-à-dire une suite de 8 bits). Le code le plus couramment utilisé se nomme ASCII (American Standard Code for Information Interchange). Bien qu'utilisant 8 bits, le code n'en comporte que 7 utiles (le premier valant toujours, soit 0, soit 1 suivant les ordinateurs). Cela fournit donc $2^7 = 128$ possibilités différentes. Parmi celles-ci, on trouve bien sûr les codes des caractères déjà rencontrés, tels que :

– les lettres de l'alphabet (majuscules et minuscules)

– les chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

– des symboles tels que : * ( ) – + = < > , / ? : $ ^ & etc...

Par ailleurs, on y trouve également des codes représentant des caractères «non affichables», comme ceux correspondant aux touches «return» ou «break» (ou CNTRL + C). Il y en a beaucoup d'autres, dont certains sont relatifs à des fonctions graphiques (déplacement de curseur, inversion vidéo, ...).

Enfin, le code ASCII laisse l'interprétation de certaines valeurs au choix du constructeur. C'est ce qui permet, sur certains matériels, de coder nos lettres avec accent, qui n'existent pas en anglo-saxon.

2.2. Le type numérique

Les nombres sont codés sur un nombre fixe de bits (généralement 32). Le code lui-même peut dépendre du système utilisé. Cependant, il s'inspire toujours du même principe qui consiste à utiliser une représentation analogue à celle employée en notation scientifique telle que :

$0.628\ 10^4$

Bien sûr, la mantisse (ici 0.628) et l'exposant (ici 4) sont exprimés en binaire. En plus, la base de l'exposant (10 dans notre exemple) est souvent de 2, parfois de 16.

Nous n'entrerons pas plus avant dans le détail. Par contre, il vous faut retenir que deux éléments essentiels interviennent dans un tel codage :

– les nombres sont représentés de manière approchée.

– il y a une limite aux valeurs que l'on peut ainsi représenter.

2.2.1. Représentation approchée

Si vous cherchez à représenter en mémoire, un nombre tel que :

0,83356727896765

il vous semble naturel de devoir vous contenter d'une certaine précision, puisque le nombre de bits utilisés est lui-même limité. Par contre, cette limitation peut paraître moins évidente pour des nombres tels que :

0,1

En effet, bien qu'il s'écrive en décimal avec un seul chiffre significatif, il est possible que sa conversion en binaire ne s'exprime plus avec un nombre limité de bits (c'est un peu comme si vous vouliez exprimer 1/3 avec un nombre fixe de décimales). Dans ces conditions, la représentation de 0,1 en mémoire sera elle aussi approchée.

A titre indicatif, disons que sur la plupart des machines, les nombres sont représentés avec une précision RELATIVE de l'ordre de 5.10^{-8}.

2.2.2. Limitation des valeurs représentées

L'exposant est lui aussi codé sur un nombre fixe de bits. Cela empêche de représenter des nombres très grands ou très petits. Sur la plupart des machines, on peut ainsi utiliser des nombres dont la valeur absolue est comprise entre :

10^{-38} et 10^{38}

(la valeur 0 étant, en outre, représentable)

REMARQUE :

Très souvent, Basic écrit les valeurs numériques avec seulement six chiffres significatifs. C'est moins que ce qu'autoriserait la précision avec laquelle les nombres sont représentés en mémoire. Dans ces conditions, on peut très bien voir deux variables dont la valeur en mémoire est différente, être écrites de manière identique comme dans cet exemple :

```
100 LET A = 1
110 LET B = 1 +.0000001
120 PRINT A;B;B-A
Ok
RUN
 1  1  1.19209E-07
Ok
```

Nous avons d'ailleurs déjà rencontré ce phénomène, notamment dans le chapitre 13, à propos de «Pythagore».

3. LES TYPES NUMÉRIQUES EN BASIC ÉTENDU

Nous avons vu qu'en Basic standard, il n'existait qu'un seul type numérique. En Basic étendu, on en trouve trois : entier, réel, double précision.

3.1. Le type réel (ou réel simple précision)

Il correspond au seul type numérique existant en Basic standard.

3.2. Le type entier

Il est utilisé pour représenter les nombres entiers relatifs. Le codage en est généralement réalisé sur 16 bits. L'un sert à préciser le signe du nombre. Les 15 autres servent à coder, en binaire, la valeur absolue du nombre (ou quelque chose de voisin). On montre qu'on peut ainsi représenter des nombres N tels que :

$$-32768 \leqslant N \leqslant 32767$$

Le principal avantage de ce type réside dans le gain de place mémoire obtenu par rapport au type réel (les variables entières occupent, en effet, deux fois moins de place que les variables réelles).

Quelques restrictions existent. Notamment, il est fréquent que les variables entières soient interdites dans les instructions FOR ou les définitions de fonctions.

Enfin un inconvénient majeur provient de ce que lorsque des variables entières apparaissent dans des expressions arithmétiques, Basic les convertit en réel, avant d'effectuer les calculs. On voit donc que cela entraîne une perte de temps à l'exécution.

3.3. Le type double précision (ou réel double précision).

Il s'agit d'un codage analogue à celui du type réel mais où l'on utilise davantage de bits pour la mantisse (la place attribuée à l'exposant restant inchangée). On montre, qu'en général la précision relative est alors de l'ordre de 5.10^{-18}

Le principal inconvénient de cette représentation est que ces variables occupent, en mémoire, plus de place que les variables réelles (en général deux fois plus).

4. ÉCRITURE DES NOMS DE VARIABLE

En Basic standard, nous avons vu qu'un nom de variable (ou de tableau) est formé d'une lettre, suivie éventuellement d'un chiffre. Les variables de type chaîne comportent, en plus, le symbole $.

En Basic étendu, vous pouvez utiliser des noms de variable ou de tableau comportant plus de deux caractères. En ce qui concerne le type, vous pouvez :

– soit le définir individuellement pour chaque variable (comme en Basic standard).

– soit le définir pour un ensemble de variables.

4.1. Écriture des noms de variable (ou de tableau).

En général, vous pouvez utiliser au moins 8 caractères alphanumériques (c'est-à-dire lettres ou chiffres). La seule restriction est que le premier caractère doit être une lettre. Ainsi, ces noms sont corrects :

XY45 RAYON SURFACE 2

Par contre, ceux-ci ne le sont pas :

4 VLR (le premier caractère n'est pas alphabétique)
CAS–3 (le – n'est pas alphanumérique)

REMARQUES IMPORTANTES :

1. Très souvent, vous aurez droit à beaucoup plus de 8 caractères (parfois jusqu'à 40 !). Cependant, il arrive que certains Basic tolèrent des noms très longs tout en ne prenant vraiment en compte que les huit premiers caractères. Dans ce cas, deux noms apparemment différents comme SURFACES1 et SURFACES2 correspondront à la même variable.

2. Certains petits systèmes autorisent des noms assez longs tout en interdisant qu'y apparaisse un «mot-clé» Basic, c'est-à-dire un mot ayant une signification précise dans le langage. Parmi ces mots, vous connaissez IF, PRINT, DATA, etc... Dans ces conditions, ces noms seraient incorrects :

EIFFEL (apparition du mot-clé IF)
COORDONNÉE (apparition du mot-clé OR)

Si votre système est de ce type, il vaut mieux vous limiter aux noms du Basic minimal, pour lesquels ce genre de phénomène ne peut pas se produire.

3. Dans tous les cas, vous n'aurez pas le droit d'utiliser un mot-clé comme nom de variable. Ceci est une restriction très sévère puisqu'elle vous oblige pratiquement à connaître tous ces mots-clés. Parmi ceux-ci, il ne faudra pas oublier les noms de fonctions comme SIN, COS, EXP, RND (et d'autres qui existent en Basic étendu comme VAL ...). Là encore, vous ne rencontrerez aucun problème si vous vous limitez aux noms du Basic standard.

4.2. Détermination du type des variables

En Basic standard, nous avons vu que les variables chaînes se reconnaissent à leur nom terminé par $. Tout autre nom est considéré comme du type numérique. En Basic étendu, nous allons retrouver ce mécanisme, mais avec cette fois quatre possibilités différentes. Nous le nommerons «déclaration explicite», car on y donne explicitement le type d'une variable. Par ailleurs, nous aurons également la possibilité de donner un type à un ensemble de variables, à l'aide d'une seule instruction que nous nommerons alors «déclaration implicite». Décrivons tout d'abord cette deuxième façon de procéder.

4.2.1. Déclaration implicite

C'est une instruction qui permet de définir le type d'un ensemble de variables (ou de tableaux). Par exemple :

```
DEFINT A
```

signifie que toutes les variables dont le nom commence par la lettre A sont de type entier.

De même :

```
DEFINT C – F
```

signifie que toutes les variables dont le nom commence par l'une des lettres C, D, E, F (C – F voulant dire de C jusqu'à F) sont de type entier.

Voici les instructions à utiliser pour chacun des quatre types :

type	instruction	
entier	DEFINT	(INTeger = entier)
réel	DEFSNG	(SiNGle precision = simple précision)
double précision	DEFDBL	(DouBLe précision = double précision)
chaîne	DEFSTR	(STRing = chaîne)

Exemple :

Si un programme comporte ces instructions :

```
100 DEFINT I-N
110 DEFSNG A-H
```

les variables I15, KILO, LONGUEUR, MX45Z seront de type entier, tandis que les variables ALPHA, CX, FREQ seront de type réel.

Vous pouvez, si vous le souhaitez, utiliser plusieurs instructions du même genre. Ainsi, avec :

```
100 DEFINT I-J
110 DEFSNG K
120 DEFINT L-N
```

J15, IMAGE et NORME seront entières, tandis que KILO et KXY seront réelles.

4.2.2. Déclaration explicite

C'était la seule possibilité qui existait en Basic standard. Elle consiste à définir le type d'une variable en faisant suivre son nom d'un symbole particulier. On utilise :

$ pour le type chaîne
% pour le type entier
! pour le type réel
pour le type double précision

Ainsi :

N% et VALEUR% sont de type entier
N# et RESX2# sont de type double précision

Ce second type de déclaration a priorité sur les déclarations implicites. Ainsi, dans ce programme :

```
100 DEFINT B-E
110 LET N % = 5
120 LET B ! = 20
130 LET D = 15
```

N% est, bien sûr entière. Mais, B! est réelle (à cause du !), malgré la présence de DEFINT B-E. Par contre, D (qui n'a pas de déclaration implicite) est soumise à la déclaration implicite, elle est donc entière.

4.2.3. En l'absence de déclaration

De même qu'en Basic standard, une variable qui ne fait l'objet d'aucune déclaration est considérée comme réelle. Ainsi, dans ce programme :

```
100 DEFDBL Y-Z
110 DEFINT I-N
120 LET Y = 25
130 LET J = 15
140 LET A = 20
```

Y est double précision (par la déclaration implicite DEFDBL)

J est entière (par la déclaration DEFINT)

A est réelle (car soumise à aucune déclaration)

5. PROBLEMES DE CONVERSION

5.1. Évaluation des expressions arithmétiques

Comme nous l'avons dit, si des variables entières apparaissent dans des expressions, elles sont converties en réel avant tout calcul. Il en va de même pour les arguments des fonctions standard. Ceci n'a pas de conséquences fondamentales sur les résultats des calculs, mais les temps d'exécution s'en trouvent accrus.

5.2. Quant on affecte un résultat à une variable entière

Exécutons ce programme :

```
100 LET N = 8
110 LET D = 3
120 LET R% = N/D
130 PRINT R%
Ok
RUN
 3
Ok
```

Que s'est-il passé ? L'expression arithmétique N/D a pour valeur approchée 2,666667. Par contre, l'instruction 120 demande d'affecter cette valeur à une variable entière R %. Il est donc nécessaire de convertir cette valeur en entier. Dans notre exemple, cela s'est fait en prenant l'entier le plus proche. Mais beaucoup de Basic procèdent en prenant la partie entière.

6. LES ERREURS ARITHMÉTIQUES

Comme nous l'avons vu, la représentation des nombres en mémoire leur impose des limites. Des problèmes apparaissent, dès que le résultat d'un calcul se trouve en dehors de ces limites.

6.1. Dépassement de capacité (Overflow).

Il se produit quand la valeur absolue d'un résultat est supérieure à la limite prévue, c'est-à-dire généralement :

* 32767 en entier
* environ 10^{38} en réel (simple ou double précision)

Basic le signale par un message du type :

```
Overflow in nnn
```

ou nnn représente le numéro de la ligne en cause.

Généralement, il remplace alors le résultat par la plus grande valeur possible et poursuit l'exécution du programme. Cependant, dans ce cas, les résultats ultérieurs peuvent devenir complètement faux !

6.2. Sous-dépassement de capacité (underflow)

Il a lieu lorsque la valeur absolue d'un nombre est inférieure à la plus petite valeur représentable, c'est-à-dire environ 10^{-38}. (Notez que cela ne peut pas arriver avec les nombres entiers).

Le résultat est alors remplacé par zéro. Certains Basic ne fournissent pas de message d'erreur. D'autres écrivent :

```
Underflow in nnn
```

Les conséquences de cette «correction autoritaire» peuvent être importantes si le résultat ainsi obtenu apparaît ensuite comme diviseur.

6.3. Division par zéro

Dès que vous cherchez à diviser un nombre quelconque (entier ou réel) par zéro, vous obtenez le message :

Division by zero in nnn

Le résultat est alors remplacé par la plus grande valeur possible, soit généralement :

* – 32767 ou + 32768 en entier

* environ ± 10^{38} en réel (simple ou double précision)

EXERCICES

1. Est-il possible de transmettre une information sans la coder ?

2. Combien de positions a-t-on utilisé dans les cas suivants :

a) 248

b) MACHINE
C) TOME1

3. Qu'est-ce que le nombre de moments d'un code ?

4. Combien d'informations différentes peut-on coder sur trois positions d'un code à 10 moments ?

5. Combien d'informations différentes peut-on coder sur deux positions d'un code à trois moments nommés a, b et c ? Écrivez toutes ces informations.

6. Qu'est-ce qu'un bit ?

7. Combien d'informations différentes peut-on coder sur :

a) 1 bit

b) 2 bits

c) 4 bits

d) 8 bits

e) 16 bits

8. Combien y-a-t-il de types en Basic standard ? Lesquels ?

9. Quel est le code le plus utilisé pour représenter les caractères ?

10. Quels sont les deux éléments essentiels intervenant dans la représentation des nombres dans le type réel ?

11. Quelle différence y-a-t-il entre le type réel et le type double précision ?

12. Quelle différence y-a-t-il entre le type réel et le type entier ?

13. Qu'est-ce que la déclaration implicite de type ?

14. Qu'est-ce que la déclaration explicite de type ?

15. Un programme possède les seules déclarations suivantes :

```
100 DEFINT I-P
110 DEFSTR X-Y
120 DEFDBL A-C
```

Quel sera le type des variables suivantes :

K   IN$ CARAC   Y7%   AZUR !   POLE   Y$

16. Qu'appelle t-on dépassement de capacité ?

17. Quel message risque de fournir l'exécution de ce programme :

```
100 LET X = 1
110 LET X = 2 * X
120 GO TO 110
```

18. Que fait Basic, en général, après avoir écrit :

Overflow in 200

19. Que fait Basic lorsqu'il rencontre une division par zéro ?

XIX
COMPLÉMENTS SUR LES CHAÎNES DE CARACTÈRES

1. INTRODUCTION

En basic standard, il y a une grande différence entre les variables de type numérique et celles de type chaîne. En effet, les premières peuvent être combinées par des opérations (+, –, *, /, ˆ) pour créer des expressions arthmétiques. On peut également leur appliquer des fonctions standard (SIN, COS, EXP, etc...) ou définies par l'utilisateur.

Par contre, avec le type chaîne, aucune manipulation de ce genre n'est possible. On doit se contenter de lire ou d'écrire de telles variables, éventuellement de transférer (par affectation) le contenu de l'une dans une autre.

En Basic étendu, de nombreuses possibilités nouvelles vont vous être offertes. S'il n'existe qu'une seule opération entre chaînes (nommée concaténation), de très nombreuses fonctions standards vont, par contre, vous permettre d'effectuer des traitements aussi variés que :

– inversion d'un mot

– remplacement d'un caractère par un autre

– conjugaison d'un verbe

– codage d'un texte (cryptage)

– etc...

2. FONCTIONS LEN, MID$, LEFT$, RIGHT$

2.1. Fonction LEN

Elle permet d'obtenir la longueur d'une chaîne, c'est-à-dire le nombre de caractères qu'elle contient. Tous les caractères sont pris en compte, y compris les blancs, mais aussi les caractères spéciaux ou certains qui n'apparaissent pas sur l'écran.

En voici un exemple, en mode direct :

```
PRINT LEN ("TOULOUSE")
8
```

Bien sûr, cette fonction ne présente pas un grand intérêt en elle-même. Ce n'est qu'en la combinant avec d'autres que nous pourrons obtenir des résultats plus attrayants.

2.2. Fonctions LEFT$, RIGHT$, MID$

Elles permettent «d'extraire» une partie (nommée souvent sous-chaîne) d'une chaîne.

2.2.1. Fonction LEFT$

Elle extrait les premiers caractères de gauche de la variable concernée. Elle possède deux arguments :

– la chaîne considérée

– le nombre de caractères à extraire.

Ainsi :

```
PRINT LEFT$ ("CECI EST UNE CHAINE", 4)
```

fournit :

```
CECI
```

Exemple 1 :

```
100 LET C1$ = "BONJOUR MADAME"
110 LET C2$ = LEFT$(C1$,7)
120 PRINT C1$;LEN(C1$)
130 PRINT C2$:LEN(C2$)
Ok
RUN
BONJOUR MADAME 14
BONJOUR 7
Ok
```

Exemple 2 :

```
100 LET C$ = "BONJOUR"
110 LET N = LEN(C$)
120 FOR I = 1 TO N
130    PRINT LEFT$(C$,I)
140 NEXT I
Ok
RUN
B
BO
BON
BONJ
BONJO
BONJOU
BONJOUR
Ok
```

2.2.2. Fonction RIGHT$

Elle joue un rôle similaire à celui de LEFT$, en portant, cette fois, sur les caractères de droite de la chaîne considérée. Ainsi :

```
PRINT RIGHT$ ("CECI EST UNE CHAINE", 10)
```

fournit :

```
UNE CHAINE
```

Exemple :

```
100 LET C$ = "BONJOUR"
110 LET N = LEN(C$)
120 FOR I = 1 TO N
130     PRINT RIGHT$(C$,I)
140 NEXT I
Ok
RUN
R
UR
OUR
JOUR
NJOUR
ONJOUR
BONJOUR
Ok
```

2.2.3. Fonction MID$

Elle extrait une sous-chaîne commençant en n'importe quel endroit d'une chaîne. Elle possède trois arguments :

– la chaîne considérée
– le rang (numéro) du premier caractère à extraire
– la longueur de la sous-chaîne à extraire

Ainsi :

MID$ ("EXEMPLE", 2,4)

extrait de la constante chaîne "EXEMPLE", quatre caractères à partir du deuxième. Nous obtenons donc la chaîne "XEMP".

Exemple :

```
100 LET C$ = "BONJOUR"
110 FOR I = 1 TO 5
120     LET E$ = MID$(C$,3,I)
130     PRINT E$
140 NEXT I
Ok
RUN
N
NJ
NJO
NJOU
NJOUR
Ok
```

REMARQUE :

LEFT$ et RIGHT$ ne sont que des cas particuliers de MID$. Ainsi :

LEFT$ (C$, I) est équivalent à MID$ (C$, 1, I)
RIGHT$ ("EXEMPLE", 3) est équivalent à MID$ ("EXEMPLE", 5,3)

Par contre, il est moins aisé de trouver l'équivalent de :

RIGHT$ (C$, I)

il faut faire intervenir la longueur de la chaîne C$ et demander que MID$ commence à extraire à partir de I caractères avant la fin. Vous pouvez vérifier qu'il faut alors écrire :

MID$ (C$, LEN (C$) – I + 1, I)

2.3. Exemple : inverser un mot

Nous souhaitons écrire un programme qui demande d'entrer une chaîne de caractères et qui la réécrit à l'envers. Pour cela, nous utilisons la fonction MID$ pour extraire un à un chacun des caractères de la chaîne, en commençant par le dernier.

```
100 INPUT C$
110 LET L = LEN(C$)
120 FOR I = L TO 1 STEP -1
130     LET C1$ = MID$(C$,I,1)
140     PRINT C1$;
150 NEXT I
Ok
RUN
? BONJOUR
RUOJNOB
Ok
```

REMARQUES :

1. Nous avons dû placer un point-virgule dans l'instruction 140, afin d'éviter le passage à la ligne à chaque écriture d'une lettre.

2. Si la variable C$ contient bien la chaîne entrée, aucune variable, par contre, ne contient la chaîne inversée. Celle-ci a seulement été écrite, au fur et à mesure qu'on obtenait chaque caractère la constituant. Nous apprendrons, dans le paragraphe suivant, comment faire pour la conserver dans une variable chaîne.

2.4. Quelques erreurs à éviter.

Si, dans ces trois fonctions LEFT$, RIGHT$ et MID$, vous demandez d'extraire un nombre négatif ou nul de caractères, vous obtiendrez une chaîne «vide» (c'est-à-dire ne contenant aucun caractère), de longueur zéro. Voici un exemple :

```
PRINT LEN (LEFT$ ("BONJOUR", – 1)
0
```

De même, si vous demandez plus de caractères qu'il n'en existe dans la chaîne considérée, vous n'obtiendrez que les caractères existant réellement, comme dans cet exemple :

```
LET C$ = "BONJOUR"
LET E$ = RIGHT$ (C$, 9)
PRINT E$, LEN (E$)
BONJOUR  7
```

REMARQUE :

Il existe certains Basic qui, dans de tels cas, fournissent un message d'erreur du type :

```
Illegal function call
```

suivi d'un arrêt de l'exécution du programme.

3. CONCATENATION DE DEUX CHAINES

3.1. L'opération «concaténation»

Il n'existe qu'une opération portant sur les chaînes de caractères. C'est la concaténation qui permet de fabriquer une chaîne en en mettant deux autres bout à bout. Elle se note + (comme l'addition numérique). Ainsi, cette instruction :

```
PRINT "BON" + "JOUR"
```

écrit :

```
BONJOUR
```

Exemple :

```
100 LET C1$ ="BONJOUR"
110 LET C2$ ="MONSIEUR"
120 LET C$ = C1$ + " " + C2$
130 PRINT C$
Ok
RUN
BONJOUR MONSIEUR
Ok
```

3.2. Inverser un mot

Reprenons l'exemple du programme 2.3, où nous avons vu une façon d'inverser les caractères d'une chaîne. Nous allons maintenant conserver dans une variable, la chaîne ainsi inversée.

```
100 INPUT C$
110 LET L = LEN(C$)
120 LET I$ = ""
130 FOR I = L TO 1 STEP -1
140     LET C1$ = MID$(C$,I,1)
150     LET I$ = I$ + C1$
160 NEXT I
170 PRINT I$
Ok
RUN
? BONJOUR
RUOJNOB
Ok
RUN
? REVER
REVER
Ok
```

L'instruction 120 place une chaîne vide (c'est-à-dire ne contenant aucun caractère) dans **I$**. A l'intérieur de la boucle FOR, chaque exécution de 140 permet d'extraire de **C$**, un des caractères que l'instruction 150 ajoute à droite de **I$** (grâce à la concaténation).

3.3. Conjuguer un verbe

Voici un programme qui demande d'entrer un verbe, vérifie qu'il est du premier groupe (c'est-à-dire, terminé par ER) et donne sa conjugaison au présent de l'indicatif.

```
100     DATA JE,E, TU,ES, IL OU ELLE,E
110     DATA NOUS,ONS, VOUS,EZ, ILS OU ELLES,ENT
120 REM
130 REM ---- BOUCLE JUSQU'A REPONSE = VERBE DU 1ER GROUPE
140         PRINT "DONNEZ UN VERBE DU PEMIER GROUPE"
150         INPUT V$
160         LET T$ = RIGHT$(V$,2)
170         IF T$ <> "ER" THEN 130
180 REM
190 REM ---- ECRITURE DE LA CONJUGUAISON
200     LET L = LEN(V$)
210     LET V$ = LEFT$(V$,L-2)
220     FOR I = 1 TO 6
230         READ S$,T$
240         LET C$ = V$ + T$
250         PRINT S$;" ";C$
260     NEXT I
Ok
RUN
DONNEZ UN VERBE DU PEMIER GROUPE
? VIVRE
DONNEZ UN VERBE DU PEMIER GROUPE
? CHANTER
JE CHANTE
TU CHANTES
IL OU ELLE CHANTE
NOUS CHANTONS
VOUS CHANTEZ
ILS OU ELLES CHANTENT
Ok
```

4.1. Fonction STRING$

Elle crée une chaîne formée de la répétition d'un même caractère. Elle comporte deux arguments :

– le nombre de fois que l'on souhaite répéter un caractère.
– le caractère à répéter.

Ainsi :

```
PRINT STRING$ (5, "A")
```

produit :

```
AAAAA
```

Exemple 1

```
100 FOR I = 1 TO 6
110     PRINT STRING$(I,"*")
120 NEXT I
Ok
RUN
*
**
***
****
*****
******
Ok
```

Exemple 2

```
100 FOR I = 1 TO 6
110     LET C$ = STRING$(I," ")
120     LET D$ = STRING$(I,"*")
130     LET E$ = C$ + D$
140     PRINT E$
150 NEXT I
Ok
RUN
 *
  **
   ***
    ****
     *****
      ******
Ok
```

4.2. Fonction SPACE$

Elle forme une chaîne contenant un certain nombre de fois le caractère blanc. C'est un cas particulier de STRING$. Ainsi :

SPACE$ (5) est équivalent à STRING$ (5," ")

5. FONCTIONS VAL ET STR$

5.1. Fonction VAL

Elle transforme, quand cela est possible, une valeur de type chaîne en une valeur numérique. Considérez ce programme :

```
100 INPUT C$
110 LET V = VAL(C$)
120 PRINT V
Ok
RUN
? 10 CM
 10
Ok
RUN
? VALEUR 28
 0
Ok
RUN
? 21.523 METRES
 21.523
Ok
RUN
? -26 M 45 CM
-26
Ok
```

D'une manière générale, VAL transforme les premiers caractères de la chaîne considérée en une valeur numérique. Le premier caractère doit être un blanc, un signe ou un chiffre. La recherche s'arrête dès que l'on rencontre un caractère qui n'est pas un chiffre (ou un

point, si c'est le premier que l'on rencontre). Comme vous l'avez vu dans l'exemple précédent, si la chaîne ne commence pas par un nombre, la valeur obtenue par VAL est alors zéro.

5.2. Fonction STR$

Elle transforme une valeur numérique en une chaîne de caractères. Voici un exemple :

```
100 INPUT V
110 LET C$ = STR$(V)
120 LET L = LEN(C$)
130 PRINT "*";C$;"*";L
Ok
RUN
? 5
* 5* 2
Ok
RUN
? -125.84
*-125.84* 7
Ok
```

Vous voyez que lorsque le nombre fourni à STR$ est négatif, le premier caractère de la chaîne obtenue est un moins. Dans le cas contraire, c'est un blanc. Par contre, contrairement à ce que l'on obtiendrait en imprimant le nombre par PRINT, il n'y a pas de blanc à la fin de la chaîne.

5.3. Application 1 : détermination du nombre de chiffres d'un nombre entier

```
100 INPUT N
110 LET N = INT(N)
120 LET N$ = STR$(N)
130 LET L = LEN(N$)
140 PRINT N;"EST UN NOMBRE DE";L-1;"CHIFFRES"
Ok
RUN
? 125
 125 EST UN NOMBRE DE 3 CHIFFRES
Ok
RUN
? 78596
 78596 EST UN NOMBRE DE 5 CHIFFRES
Ok
```

5.4. Application 2

Ayant un nombre entier quelconque, on souhaite l'écrire sur sept caractères, en remplacant tous les blancs par des «*». Ainsi :

45 s'écrirait ***** 45
7421 s'écrirait *** 7421

Le programme pourrait être :

```
100      INPUT N
110      LET N = INT(N)
120 REM ---- CREATION DE LA CHAINE CORRESPONDANT A N
130      LET N$ = STR$(N)
140      LET L = LEN(N$)
150 REM ---- ELIMINATION DU BLANC DE GAUCHE S'IL Y A LIEU
160      IF LEFT$(N$,1) <> " " THEN 200
170         LET N$ = RIGHT$(N$,L-1)
180         LET L = L-1
190 REM ---- ECRITURE
200      PRINT STRING$(7-L,"*");N$
Ok
RUN
? 123
****123
Ok
```

```
RUN
? -43256
*-43256
Ok
RUN
? +12
*****12
Ok
```

REMARQUE :

Ce programme peut facilement se généraliser au cas où l'on souhaite écrire, sous cette même forme, un montant en francs, arrondi au centime le plus proche.

6. FONCTIONS ASC ET CHR$

6.1. Retour sur le codage des caractères

Comme nous l'avons dit, le code le plus utilisé est le code ASCII. Chaque caractère y est codé sur 8 bits, dont le premier est généralement inutilisé (il est toujours soit à 0, soit à 1 suivant les constructeurs). Par exemple, le code de la lettre C (majuscule) sera :

01000011 ou 11000011

La fonction ASC vous permet de connaître le code d'un caractère, tandis que CHR$ vous produit un caractère dont vous fournissez le code. Mais, rassurez-vous. Vous n'aurez pas besoin de manipuler le code binaire. En effet, vous utiliserez la «valeur décimale» correspondante. Ainsi, le premier code de la lettre C peut être considéré comme l'écriture en base 2 du nombre décimal 67. Dans ces conditions, ASC («C») vaudra 67 tandis que CHR$ (67) sera la chaîne «C»

D'une manière générale, à chaque caractère correspond un nombre compris entre 0 et 127 (ou 128 et 255 sur certaines machines) et réciproquement.

6.2. La fonction ASC

Voici quelques exemples en mode direct :

```
PRINT ASC ("A") ; ASC ("C") ; ASC ("X")
65 67 88
```

Si vous appliquez ASC à une chaîne de plus d'un caractère, seul le premier sera pris en compte. Par exemple :

```
PRINT ASC ("BONJOUR") ; ASC ("BABA")
66 66
```

6.3. La fonction CHR$

Voici un petit programme écrivant les 26 lettres de l'alphabet :

```
100 FOR I = 65 TO 90
110     PRINT CHR$(I);
120 NEXT I
Ok
RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

Vous pouvez utiliser CHR$ pour fabriquer ou pour envoyer à l'écran un caractère que vous ne pouvez pas entrer au clavier, soit parce qu'il n'existe pas, soit parce que sa frappe provoque un effet non désiré comme :

– effacement d'écran
– tabulation, déplacement de curseur
– inversion vidéo
– touche return, break, ...
– etc...

6.4. Application : cryptage d'un texte

Voici un programme qui «crypte» un texte, en remplaçant chaque lettre par sa suivante dans l'alphabet. Ainsi A est remplacé par B, H est remplacé par I. Par contre, Z sera remplacé par A. Pour cela, nous nous servons du fait que les valeurs correspondant aux codes des 26 lettres sont consécutives. Ainsi A vaut 66, B vaut 67,

C vaut 68, et ainsi de suite. Par ailleurs, nous supposons que le texte à crypter ne contient que des lettres (et en outre majuscules).

```
100      LET CRYP$ = ""
110      INPUT TEXTE$
120      FOR I = 1 TO LEN(TEXTE$)
130 REM ---- PRELEVEMENT DANS C$ DU CARACTERE DE RANG I
140         LET C$ = MID$(TEXTE$,I,1)
150 REM     ---- EST-CE UN Z ?
160         IF C$ = "Z" THEN 210
170 REM     ---- NON - ON CONCATENE SON SUIVANT A CRYPT$
180            LET V = ASC(C$)
190            LET CRYP$ = CRYP$ + CHR$(V+1)
200            GOTO 230
210 REM     ---- OUI - ON CONCATENE A A CRYPT$
220            LET CRYP$ = CRYP$ + "A"
230      NEXT I
240      PRINT CRYP$
Ok
RUN
? ANTICONSTITUTIONNELLEMENT
BOUJDPOTUJUVUJPOOFMMFNFOU
Ok
RUN
? ZOO
APP
Ok
```

EXERCICES

1. Quel est le rôle de la fonction LEN ?

2. Que produira ce programme :

```
100 INPUT C$
110 PRINT LEN (C$)
```

si vous répondez :

a) PARIS

b) "1.2.OU.TROIS"

c) " "

3. A quoi servent les fonctions LEFT$ et RIGHT$?

4. Que produira ce programme :

```
100 INPUT C$
110 FOR I = 1 TO LEN (C$)
120 PRINT LEFT$ (C$, I) ; RIGHT (C$, I)
130 NEXT I
```

si vous répondez :

a) MONSIEUR

b) REVER

5. Écrire un programme qui demande d'entrer une chaîne et qui la réécrit, à raison d'un caractère par ligne. Ainsi, si vous répondez JOUR, le programme devra vous fournir :

```
J
O
U
R
```

6. Qu'obtiendrez vous en exécutant ce programme :

```
100 LET C$ = "MADEMOISELLE"
110 LET C1$ = LEFT$ (C$, 0)
120 LET C2$ = MID$ (C$, 9, 10)
130 PRINT C1$ ; C2$
```

7. Qu'est-ce que la concaténation ?

8. Que produira ce programme :

```
100 INPUT C$
110 INPUT D$
120 LET R$ = "MIEUX VAUT "+ C$+" QUE" + D$
130 PRINT R$
```

si vous répondez :

RIRE
PLEURER

9. A quoi servent les fonctions STRING$ et SPACE$?

10. Quelle différence y-a-t-il entre :

ASC ("O") et ASC ("OUI")

11. Qu'obtiendra-t-on en frappant :

a) PRINT CHR$ (ASC ("B"))
b) PRINT CHR$ (ASC ("BONJOUR"))

12. Que fournira ce programme :

```
100 INPUT C$
110 PRINT CHR$ (ASC (C$) + 2)
```

si l'on répond :

a) D

b) MONSIEUR

XX
COMPLÉMENTS SUR IF

Le basic étendu vous offre deux nouvelles possibilités :

– l'utilisation d'expressions logiques rendant plus aisée l'écriture de conditions complexes.

– le IF dit généralisé dans lequel vous pouvez placer n'importe quelle instruction Basic (et non plus seulement un branchement).

En outre, sur certains systèmes, vous trouverez un «IF structuré». Celui-ci permet d'exprimer simplement une structure de choix.

1. LES EXPRESSIONS LOGIQUES

1.1. Introduction

En basic standard, vous pouviez écrire :

```
50 IF A < B THEN 100
```

En basic étendu, il vous sera possible d'utiliser des expressions plus compliquées telles que :

```
IF A < B AND C > = 5 THEN 100
```

Dans les deux cas, la condition placée après le IF porte le nom d'expression logique. Une telle expression ne peut être que vraie ou fausse.

Les expressions simples telles que A < B, qui étaient les seules autorisées en basic standard, s'appellent «expressions de relation». Ce sont des expressions logiques particulières formées de la comparaison entre deux expressions arithmétiques ou entre deux chaînes de caractères.

1.2. Les expressions logiques en basic étendu

Bien sûr, vous avez toujours la possibilité d'utiliser de simples expressions de relation. Mais, en outre, vous pouvez écrire des combinaisons d'expressions de relation à l'aide des «opérateurs logiques» :

AND OR NOT

Ainsi :

C = 3 AND X = 4 sera vraie si à la fois C vaut 3 et X vaut 4.

C = 3 OR X = 5 sera vraie si C vaut 3 ou bien si X vaut 5 (ou si ces deux conditions sont vraies en même temps)

NOT C = 3 sera vraie si C = 3 est faux. Ici, cela est équivalent à C < > 3

D'une manière générale, vous aurez fréquemment le droit d'écrire des expressions plus compliquées contenant plusieurs opérateurs logiques et éventuellement des parenthèses (pour imposer des priorités). En pratique, leur utilisation est peu courante. A titre d'exemple, disons que :

(C = 3 AND X = 4) OR (K = 1 AND J < K)

est vraie si au moins une de ces deux affirmations est vraie :

C = 3 AND X = 4
K = 1 AND J < K

1.3. Applications

– Ce type d'expression logique permet de simplifier certaines écritures. Ainsi, l'instruction :

```
100 IF A < B AND A < C THEN 250
```

remplace avantageusement, et de façon plus lisible :

```
100 IF A > = B THEN 130
110 IF A > = C THEN 130
120 GO TO 250
130 REM
```

– La demande d'une réponse de type OUI, NON avec formulation de la question jusqu'à réponse correcte, s'écrit simplement :

```
100 PRINT "DONNEZ VOTRE REPONSE (OUI OU NON)"
110 INPUT R$
120 IF R$ <> "OUI" AND R$ <> "NON" THEN 100
```

2. IF GENERALISE

2.1. Introduction

En Basic standard, vous ne pouviez trouver après THEN qu'un numéro de ligne auquel Basic se branchait lorsque l'expression logique était vraie.

En Basic étendu, vous pouvez écrire :

```
100 IF A < B THEN PRINT "OK"
```

qui signifie : si A < B écrire «OK». Puis, que cette condition soit vraie ou non, Basic exécute l'instruction suivante.

De même :

```
100 IF I < N THEN LET I = I + 1
```

augmente de 1 la valeur de I si I < N

D'une manière générale, vous pouvez placer après THEN n'importe quelle instruction, sauf FOR.

Vous noterez que, si cette instruction est un GO TO, vous retrouvez la fonction du IF du Basic standard. Ainsi :

```
50 IF N > 5 THEN GO TO 100
```

est équivalent à :

```
50 IF N > 5 THEN 100
```

2.2. Applications

– Là encore, l'utilisation de cette nouvelle forme du IF permet de simplifier certaines écritures. Ainsi :

```
100 IF I > N THEN LET I = N
```

remplace :

```
100 IF I <= N THEN 120
110 LET I = N
120 REM
```

– Voici un exemple de recherche, dans R, de la valeur maximum de deux variables A et B :

```
100 IF A < B THEN LET R = B
110 IF A >= B THEN LET R = A
```

Notez qu'il faut prendre garde à ne pas écrire :

```
110 IF A > B THEN LET R = A
```

car alors, quand A serait égal à B, la variable R ne serait pas définie.

Vous pouvez également écrire :

```
100 LET R = A
110 IF B > A THEN LET R = B
```

3. IF STRUCTURE

Certains Basic vous permettent des instructions telles que :

```
200 IF A > B THEN LET R = A ELSE LET R = B
```

qui signifie :

si A > B alors exécuter LET R = A
sinon exécuter LET R = B

D'une manière générale, cette instruction s'écrit :

IF expl THEN inst1 ELSE inst2

où :

expl est une expression logique
inst1 et inst2 sont des instructions Basic quelconques.

L'intérêt du IF structuré est de décrire une structure de choix, à l'aide d'une seule instruction Basic. Cependant, il faut que chaque partie du choix s'exprime par une seule instruction.

De nombreux systèmes autorisent plusieurs instructions sur une même ligne (séparées par :). Dans ces conditions, la restriction énoncée ci-dessus n'est plus vraie. Encore faut-il néanmoins que l'instruction IF complète tienne en une seule ligne (dans certains cas, vous aurez droit à des lignes de 255 caractères).

XXI

AUTRES POSSIBILITÉS DU BASIC ÉTENDU

Nous n'avons pas voulu charger ce manuel d'initiation en décrivant systématiquement en détail toutes les possibilités du basic étendu. Ce dernier chapitre a pour objet de vous mentionner celles que nous n'avons pas encore abordées et qui existent sur la plupart des micro-ordinateurs.

1. INPUT "CHAINE"

Il est très fréquent que vous fassiez précéder une instruction INPUT d'une instruction PRINT pour préciser quelle réponse vous en attendez. C'est le cas dans cet exemple :

```
100 PRINT "DONNEZ UN NOMBRE ENTIER"
110 INPUT N
```

En Basic étendu, vous pouvez résumer cela en une seule instruction :

```
100 INPUT "DONNEZ UN NOMBRE ENTIER" ; N
```

L'exécution de cette instruction provoque l'écriture du texte mentionné entre guillemets. Puis Basic imprime, sans passer à la ligne, un point d'interrogation et attend que vous lui fournissiez une valeur pour N.

Notez bien que vous ne pouvez placer qu'un seul texte et que celui-ci doit toujours être placé avant la liste des variables. Voici un autre exemple :

```
100 INPUT "DONNEZ VOTRE NOM ET VOTRE AGE" ; NOM$, AGE
```

2. PRINT USING

Si vous souhaitez aligner des résultats, TAB vous permet de le faire, du moins dans une certaine mesure. Examinez cet exemple :

```
100 DATA 0.5,1,10,152.3
110 FOR I = 1 TO 4
120    READ X
130    PRINT X
140 NEXT I
Ok
RUN
 .5
 1
 10
 152.3
Ok
```

Si vous désirez que les points décimaux de vos nombres soient alignés, comme ceci :

```
  0.5
  1
 10
152.3
```

il vous faut utiliser une nouvelle possibilité : PRINT USING. Elle vous permet de spécifier le «format» ou le «cadrage» des valeurs à imprimer.

Nous n'entrerons pas dans le détail de cette nouvelle instruction qui peut prendre différents aspects suivant les ordinateurs utilisés. Nous préférons vous donner ici quelques exemples qui vous en feront lement saisir l'intérêt :

```
100 LET A = 123.456
110 PRINT USING "+###.###";A
120 PRINT USING "+####.###";A
130 PRINT USING "+#.###^^^^";A
140 LET FORMAT$ = "####.###+"
150 PRINT USING FORMAT$;A
Ok
RUN
+123.456
 +123.456
+1.235E+02
 123.456+
Ok
```

3. PEEK ET POKE

Sur beaucoup de micro-ordinateurs, PEEK et POKE vous donnent, sous une forme certes rudimentaire, des moyens de communiquer avec le «système». PEEK vous permet de connaître le contenu de n'importe quel emplacement de la mémoire. POKE place une valeur donnée en l'emplacement mémoire de votre choix.

Pour utiliser PEEK et POKE, il vous faut savoir, que, dans la mémoire, chaque emplacement de 8 bits (nommé «octet») est repéré par un nombre qu'on appelle son «adresse».

3.1. Fonction PEEK

Elle s'utilise dans n'importe quelle expression arithmétique. PEEK (A) fournit la valeur décimale de l'octet situé à l'adresse A.

Vous pouvez ainsi examiner toute la mémoire de votre ordinateur. Notamment, en consultant certains octets d'adresse bien définie, vous obtiendrez des informations sur :

– la position du «curseur» sur l'écran.

– la touche qui vient d'être enfoncée (sans avoir à attendre que l'ensemble d'un message soit transmis par «return»), etc...

D'une manière générale, le manuel constructeur vous précisera les adresses précises de tous ces octets ayant une signification particulière.

3.2. Instruction POKE

Contrairement à PEEK qui est une fonction, POKE est une instruction. Elle place une valeur donnée (comprise entre 0 et 255) dans un octet d'adresse donnée.

Son principal intérêt réside dans le fait qu'elle vous permet de communiquer au système des informations telles que :

– inversion vidéo

– modification du jeu de caractères d'écran

– passage en «mode graphique»

– nombre de lignes par page d'imprimante

– clignotement

– etc...

Là encore, le manuel constructeur vous donnera de plus amples informations.

Nous vous conseillons vivement de vous limiter au maximum dans l'utilisation de POKE. En effet, vous risquez, par mégarde, de modifier n'importe quelle partie de la mémoire (du moins, lorsque votre système ne se trouve pas dans une mémoire protégée du type ROM). Dans ces conditions, tout comportement anormal de votre système devient possible :

– programme interrompu

– programme irrécupérable

– dialogue avec basic impossible

– envoi intempestif d'information sur écran, sur disquette.

– etc...

4. COMMANDES AUTO ET RENUM

4.1. AUTO

Nous avons appris à créer un programme, en fournissant un numéro de ligne pour chaque instruction. La commande AUTO demande à Basic de vous fournir «automatiquement» des numéros de ligne. Ceux-ci s'inscrivent alors sur l'écran, au fur et à mesure

des besoins, et vous n'avez qu'à taper les instructions correspondantes. Par exemple :

```
AUTO 100, 10
```

vous permet d'entrer des instructions, en commançant au numéro 100, et en augmentant à chaque fois de 10 le numéro de ligne.

Pour quitter ce mode (qui vous fournit sans cesse un nouveau numéro pour l'instruction suivante), il vous suffit, en général, de taper «break» (ou control + C).

4.2. RENUM

Il est fréquent, qu'après plusieurs modifications, plusieurs instructions portent des numéros consécutifs, en ce qui peut vous empêcher d'en insérer une nouvelle. La commande RENUM permet de «renuméroter» toutes les lignes. Ainsi :

```
RENUM 100, 3, 10
```

demande de renuméroter les lignes à partir de celle portant actuellement le numéro 3, en lui attribuant le numéro 100, et en augmentant les numéros de 10 à chaque ligne.

Par défaut, RENUM 100 renumérote toutes les lignes, de 10 en 10, en attribuant le numéro 100 à la première.

L'intérêt de cette commande réside dans le fait qu'elle ne se contente pas de modifier les seuls numéros de ligne. Chaque instruction où apparaît un numéro (IF, GO TO) est également corrigée en conséquence.

En voici un exemple :

```
LIST
3     LET S1 = 0
5     LET S2 = 0
10    FOR I = 1 TO 10
11       INPUT X
12       IF X < 0 THEN 20
15          LET S1 = S1 - X
16          GOTO 30
20 REM
25          LET S2 = S2 + X
30 REM
```

```
32     NEXT I
Ok
RENUM 100
Ok
LIST
100      LET S1 = 0
110      LET S2 = 0
120     FOR I = 1 TO 10
130        INPUT X
140        IF X < 0 THEN 170
150           LET S1 = S1 - X
160           GOTO 190
170 REM
180           LET S2 = S2 + X
190 REM
200     NEXT I
Ok
```

REMARQUE :

L'écriture de ces commandes AUTO et RENUM peut différer suivant les systèmes.

5. AFFECTATION GÉNÉRALISÉE

La plupart du temps, le mot LET peut être omis dans l'instruction d'affectation.

Ainsi :

```
100 A = B + C
```

est équivalent à :

```
100 LET A = B + C
```

6. PLUSIEURS INSTRUCTIONS SUR UNE MEME LIGNE

Nous en avons déjà parlé au chapitre 20. Chaque instruction est séparée de la précédente par deux-points (:)

Ainsi :

```
100 FOR I = 1 TO 10 : PRINT I : NEXT I
```

remplace ces trois instructions :

```
100 FOR I = 1 TO 10
110 PRINT I
120 NEXT I
```

Vous pouvez ainsi ajouter des remarques sur la même ligne qu'une instruction, comme dans cet exemple :

```
100 IF A < B THEN R = A ELSE R = B : REM MAX DE A ET B DANS R
```

Cependant, généralement, vous ne pouvez pas placer d'autres instructions après cette remarque car, tout ce qui suit le mot REM (y compris d'éventuels :) est considéré comme commentaire. Ainsi, dans :

```
100 X = 1 : REM INIT COMPTEUR : S = 0
```

il n'y a que deux instructions :

* une affectation : X = 1
* une remarque : REM INIT COMPTEUR : S = 0

7. INSTRUCTION STOP ET COMMANDE CONT.

L'instruction STOP permet d'interrompre temporairement l'exécution d'un programme. Par exemple, dans :

```
    -----
190 STOP
200 -----
    -----
```

lorsque Basic rencontre l'instruction 190, il vous donne un message du type :

```
Break in 190
```

Vous pouvez alors exécuter toutes les commandes que vous désirez ou des instructions en mode direct. (Par exemple, pour examiner ou modifier certaines variables).

Il vous suffit ensuite de taper la commande :

```
CONT
```

pour que Basic poursuive l'exécution du programme à partir de l'instruction suivant le STOP (ici 200).

Vous noterez que STOP produit un effet semblable à celui obtenu par «break». Toutefois, dans le premier cas, l'arrêt est prévu dans le programme. Dans le second cas, la décision d'interrompre le programme est prise durant son exécution. En général, vous ne pouvez alors prévoir avec précision l'endroit où Basic s'arrêtera.

CORRIGÉS DES EXERCICES

Note :

Lorsque l'exercice proposé comporte l'écriture d'un programme, les réponses fournies ici doivent être considérées comme un exemple de solution. Généralement beaucoup d'autres rédactions seront possibles.

Chapitre I

I.1. Le clavier et l'écran (éventuellement l'imprimante)

I.2. La mémoire centrale et l'unité centrale

I.3. Des instructions de programme en langage machine et des données.

I.4. C'est l'ensemble des instructions que l'unité centrale est capable d'exécuter.

I.5. Faux (elle ne peut exécuter que des instructions du langage machine).

I.6. La compilation et l'interprétation

I.7. Dans le cas de l'interprétation.

I.8. Le moniteur, l'interprêteur Basic et votre programme.

I.9. A transmettre un message que l'on vient de frapper.

Chapitre II

II.1. PRINT 1.39 + 4.734

II.2. PRINT 1.39 * 4.734

II.3. PRINT 41/11

II.4. 6

II.5. 1

II.6. 1

II.7. 20

II.8. LET C = 5

II.9. LET A = 5
LET B = A ^2

II.10. 3

II.11. 8

II.12. 1.5

II.13. 4 6.5

II.14. 2.5 10

Chapitre III

III.1. PRINT 5 + 3 est une instruction en mode direct. Elle sera exécutée immédiatement par Basic qui écrira :

8

20 PRINT 5 + 3 est une instruction qui sera enregistrée par Basic. Elle ne sera exécutée que lorsqu'on exécutera le programme auquel elle appartient.

III.2. en tapant simplement son numéro de ligne, suivi de «return».

III.3. On tape l'instruction corrigée :

20 LET B = 2 * A

III.4. On remplace l'instruction 30 par :

30 PRINT A, B

III.5. Il écrit : 2

III.6. L'instruction INPUT permet d'introduire des valeurs au moment de l'exécution d'un programme.

III.7. Cela signifie : demander, par une instruction INPUT, d'introduire une valeur au clavier.

III.8. a) faux b) vrai

III.9. Non. Par contre, on pourra écrire :

LET X = 5

ou encore :

INPUT X

III.10. A ce qu'il écrit un point d'interrogation.

III.11. 10 INPUT A
20 LET B = 2 * A
30 PRINT B

III.12. Rien, car Basic se contente d'enregistrer cette instruction (bien qu'elle comporte une erreur). Ce n'est qu'en exécutant le programme correspondant que l'on obtiendra le message :

Syntax error in 10

III.13. Syntax error in 10

III.14. A faciliter la modification des instructions d'un programme.

III.15. Taper «return» pour quitter ce mode.

III.16. 10 INPUT X
20 PRINT 2 * X

III.17. 10 INPUT X
20 PRINT X ^3

III.18. a) vrai b) vrai c) vrai d) vrai

Chapitre IV

IV.1. Les commandes sont : LIST SAVE"PROG1" LOAD"PROG2" NEW RUN

IV.2. A effacer le programme se trouvant en mémoire.

IV.3. 20 LET B = A + 4

IV.4. A conserver un programme dans une mémoire permanente (cassette, disquette).

IV.5. A recopier en mémoire un programme conservé dans une mémoire permanente.

IV.6. A faire des «remarques» dans un programme pour en faciliter la lecture et la compréhension.

IV.7. Lorsque l'on fait une liste du programme.

IV.8. Rien. Il passe à l'instruction suivante.

IV.9. Aucune

IV.10. A =   5

IV.11. DONNEZ MOI UN NOMBRE
? 5
VOICI SON CUBE :   125

IV-12. Oui, bien que les messages écrits ne correspondent pas à ce qui est réellement fait par le programme.

IV.13. On obtiendra :

S

et non pas la valeur de S. Pour cela, il aurait fallu écrire, par exemple :

30 PRINT "S =", S

Chapitre V

V.1. a) faux b) faux c) vrai d) faux e) faux f) faux g) vrai

V.2. L1 R7 Z

V.3. a) faux b) vrai c) faux

V.4. Oui (elle augmente de 2 la valeur de N).

V.5. Elle effectue le calcul exprimé par A * X + 3 et elle range le résultat dans C. (On dit aussi qu'elle affecte à C la valeur de l'expression arithmétique A * X + 3).

V.6. Elle est incorrecte

V.7. Seule a est correcte.

V.8.

	F5	G
10 LET F5 = 4	4	–
20 LET G = F5 – 3	4	1
30 LET F5 = G	1	1

V.9.

	X	Y
10 LET X = 4	4	–
20 LET Y = 10	4	10
30 LET X = Y	10	10
40 LET Y = X	10	10

V.10.

	X	Y	W
10 LET X = 3	3	–	–
20 LET Y = 10	3	10	–
30 LET W = X	3	10	3
40 LET X = Y	10	10	3
50 LET Y = W	10	3	3

Les instructions 30 à 50 échangent le contenu des variables X et Y.

V.11. 9 10

V.12. Non. Elle intervient seulement dans l'écriture de certaines instructions Basic.

V.13. Dans une instruction LET ou PRINT

V.14. Faux.

V.15. 10 – il manque un symbole opératoire entre 3 et B. (c'est probablement *).

20 – il manque un symbole opératoire entre 3 et (. (c'est probablement *).

30 – A * B n'est pas un nom de variable.

40 — Il manque le nom de la variable à laquelle on veut affecter la valeur de l'expression arithmétique, ainsi que le signe =

50 — AB n'est pas un nom de variable correct (ou il manque un symbole opératoire entre A et B).

60 — Y = 15 n'est pas une expression arithmétique.

70 — correcte

80 — correcte

90 — A . X n'est pas un nom de variable correct, ni une expression arithmétique. (On a probablement voulu écrire PRINT A * X).

95 — correcte

V.16. C'est une variable que l'on veut utiliser alors qu'elle n'a pas encore reçu de valeur.

V.17.
```
10 PRINT "DONNEZ LE RAYON"
20 INPUT R
30 LET V = 4/3 * 3.14159 * R ^3
40 PRINT "VOLUME =", V
```

Chapitre VI

VI.1. a) faux b) faux

VI.2. 9 est une constante numérique, tandis que "ERREUR" est une constante chaîne de caractères.

VI.3. 25 est une constante numérique, tandis que "25" est une constante chaîne de caractères.

VI.4. a) Z n'est pas une variable chaîne. On pourrait utiliser Z$.

b) 35 n'est pas une constante chaîne, alors que X$ est une variable chaîne. On peut écrire :

soit LET X = 35
soit LET X$ = "35"

c) A est une variable numérique alors que "3" est une constante chaîne.

d) correcte

e) "2 HEURES" est bien une constante chaîne, mais "2 HEURES" — 5 n'en est plus une.

f) On ne peut effectuer d'opérations entre variables chaînes. H$ — C$ est donc incorrect.

VI.5. On obtiendra, dans les deux cas :

```
DATE          30       JANVIER
```

VI.6. Non

VI.7. a) rien b) rien c) rien

d) On aura le message :

Redo from start

suivi d'un nouveau point d'interrogation, car la réponse fournie est une chaîne de caractères alors que INPUT A attendait une valeur numérique.

VI.8. Ici le programme attend une chaîne de caractères. Toutes les réponses sont donc correctes (bien que non placées entre guillemets).

VI.9. Il faut, comme dans l'exercice V.10, utiliser une troisième variable :

```
50 LET C$ = A$
60 LET A$ = B$
70 LET B$ = C$
```

VI.10. Dans le premier cas, les chaînes "BONJOUR" et "MONSIEUR" sont séparées par des blancs (la première commence à la première colonne, tandis que la seconde commence à la dix-septième colonne). Dans le second cas, elles sont accolées.

VI.11. Aucune différence. On obtient, dans les deux cas :

BONJOUR . MONSIEUR

VI.12. – 315 ˆ+ ˆ315 ˆ= ˆ0   (ˆ désignant un espace)

VI.13. a et b produiront le message :

Redo from start

(car la réponse fournie n'est pas numérique)

c fournira :

SURFACE = 314.159

Chapitre VII

VII.1. SI A est inférieur à 20 aller à l'instruction numéro 50, sinon passer à l'instruction suivante.

VII.2. a) faux b) faux

VII.3. Il écrit toujours 10, quelle que soit la valeur que l'on fournit pour N.

VII.4. On peut comparer soit des chaînes de caractères, soit des expressions arithmétiques.

VII.5. 10 – correcte

20 – C$ est une variable chaîne de caractères, alors que Z est une variable numérique. Leur comparaison est impossible.

30 – Y n'est pas un numéro de ligne.

40 – X – 8 = Y ne doit pas être entre parenthèses.

50 – On ne peut pas comparer trois expressions, mais seulement deux.

VII.6. faux

VII.7 PARIS EST AVANT TOULOUSE
TOUS EST AVANT TOUTES
TOUT EST AVANT TOUTES
^ZOE EST AVANT ANNIE

(car le caractère blanc, noté ^ est avant le A)

A3 EST AVANT A5
A25 EST AVANT A9 (car 2 est avant 9)
6 EST AVANT B
67 EST AVANT 95
48 EST APRES 145 (car 4 est après 1)

VII.8.

```
100 PRINT "DONNEZ DEUX NOMBRES"
110 INPUT A, B
120 IF A >= B THEN 150
130 PRINT "LE PLUS GRAND EST" ; B
140 GO TO 160
150 PRINT "LE PLUS GRAND EST" ; A
160 END
```

VII.9.

```
100 PRINT "QUEL EST VOTRE NOM"
110 INPUT N$
120 PRINT "ETES-VOUS UN HOMME (OUI OU NON)"
130 INPUT R$
140 IF R$ = "OUI" THEN 170
150 PRINT "BONJOUR MADAME" ; N$
160 GO TO 180
170 PRINT "BONJOUR MONSIEUR" ; N$
180 REM
```

Chapitre VIII

VIII.1. C'est une variable qui sert à compter le nombre de tours d'une boucle.

VIII.2. L'initialisation, l'incrémentation et le test d'arrêt.

VIII.3. La première manière consiste à utiliser un compteur que l'on initialise, incrémente et teste.

La seconde manière consiste à utiliser l'instruction FOR.

Application :

Première manière :

```
100 LET I = 4
110 IF I > 25 THEN 150
120     PRINT I
130     LET I = I + 1
```

```
140        GO TO 110
150 REM
```

Deuxième manière :

```
100 FOR I = 4 TO 25
110        PRINT  I
120 NEXT I
```

VIII.4. a) 4 tours N = 1, 2, 3, 4
b) 3 tours N = 6, 7, 8
c) 1 tour N = 10

VIII.5.

```
COMPTEUR DANS LA BOUCLE = 1
COMPTEUR DANS LA BOUCLE = 2
COMPTEUR DANS LA BOUCLE = 3
COMPTEUR DANS LA BOUCLE = 4
COMPTEUR EN SORTIE DE BOUCLE = 5
```

VIII.6.

```
COMPTEUR DANS LA BOUCLE = 1
COMPTEUR DANS LA BOUCLE = 5
COMPTEUR EN SORTIE DE BOUCLE = 9
```

VIII.7. 22

VIII.8.

```
100 LET S = 0
110 FOR I = 1 TO 15
120        LET S = S + I ^2
130 NEXT I
140 PRINT S
```

VIII.9.

```
100 LET S = 0
110 FOR I = 1 TO 17 STEP 2
120        LET S = S + I ^3
130 NEXT I
140 PRINT S
```

VIII.10. a) 21 tours X = 0.1, 0.2, 0.3, . . . 1.9, 2 (parfois 20 tours seulement – voir & 4.2)
b) 3 tours X = 1, 1.5, 2
c) 1 tour X = 1
d) 6 tours X = 5, 4, 3, 2, 1, 0
e) 11 tours X = 1, 0.9, 0.8, . . . 0.1, 0
(parfois 10 tours seulement – voir & 4.2)

VIII.11.

```
100 FOR X = 0 TO 2 STEP 0.05
110     LET Y = (X ^2 + 2 * X + 3)/(X + 5)
120     PRINT X, Y
130 NEXT X
```

REMARQUE :

Sur certaines machines, on aurait intérêt à écrire, par exemple :

```
100 FOR X = 0 TO 2.0001 STEP 0.05
```

pour être certain d'atteindre la valeur 2.

VIII.12. faux

VIII.13. Le compteur est modifié dans la boucle.

VIII.14. La variable mentionnée dans le NEXT ne correspond pas à celle utilisée dans le FOR. (On aura, à l'exécution, un message du type :

Next without for in 120

VIII.15. En utilisant une touche «Break» ou «Interupt» (ou Control + C)

Chapitre IX

IX.1.
```
1  10
1  20
2  10
2  20
3  10
3  20
```

IX.2.
```
1
2
3
10
20
```

IX.3. a) 10 fois b) 50 fois c) 10 fois

IX.4. Les boucles FOR X et FOR Y sont mal imbriquées. Il faudrait échanger les instructions NEXT X et NEXT Y.

IX.5. faux

IX.6.

```
100 FOR I = 1 TO 9
110     PRINT "TABLE DES" ; I
120     FOR J = 1 TO 9
130         PRINT I ; "X" ; J ; "=" ; I * J
140     NEXT J
150 NEXT I
```

IX.7.

```
100 FOR X = 0 TO 1 STEP 0.1
110     FOR Y = 0 TO 1 STEP 0.05
120         LET Z = 2 * X ^2 + 4 * X * Y + 3 * Y ^2
130         PRINT X ; Y ; Z
140     NEXT Y
150 NEXT X
```

Chapitre X

X.1. a) vrai b) faux c) faux d) vrai e) faux g) faux h) faux

X.2. 1
5 10

X.3. la même chose que ci-dessus

X.5. 1 5
Out of DATA in 130

X.5. 10 50 80
10 ^^50 ^^80 (^ désignant un blanc)

X.6.

```
DUPONT          A               15              ANS

DUPONT^A^15^ANS             (^ désignant un blanc)
```

X.7. Lorsque l'on souhaite y placer des chaînes de caractères contenant des blancs ou des virgules.

X.8. Dans le premier, on affecte la valeur numérique 128 à la variable N alors que dans le second, on affecte la chaîne 128 à la variable N$.

X.9. On obtiendra :

Syntax error in 120

X.10. Le premier fournit :

VALEUR = ^155 (^ signifie blanc)

Le second fournit :

VALEUR = 155 (sans aucun blanc)

X.11. 120 PRINT "MONSIEUR ^" ; N$;"^HABITE ^A^"; V$

(^ désignant le caractère blanc)

X.12. Non, car PRINT N imprime un blanc après 16.

X.13. A demander à Basic de reprendre la lecture des DATA depuis le début.

X.14. 10 20
10
20
30
40

X.15.

```
XXXXX
X
X
XXX
X
X
XXXXX
```

Chapitre XI

XI-1. C'est un ensemble d'emplacements mémoire, de même type, repérés par un nom unique.

XI.2. Comme les noms de variable (c'est-à-dire une lettre ou une lettre suivie d'un chiffre — le tout étant terminé par $ s'il s'agit d'un tableau de chaînes de caractères).

XI.3. C'est une expression arithmétique dont la valeur sert à préciser l'emplacement d'un élément dans un tableau.

XI.4. C'est un élément d'un tableau.

XI.5. Un indice peut être une constante, une variable ou n'importe quelle expression arithmétique Basic.

XI.6. A demander à Basic de réserver l'emplacement d'un tableau.

XI.7. a) faux

b) faux (elles doivent être comprises entre 1 et la dimension du tableau)

c) pas nécessairement. Elle doit seulement apparaître avant la première utilisation du tableau correspondant.

d) faux (voir c) e) vrai

f) faux g) vrai

XI.8. a) Elles réservent les emplacements :

— d'un tableau C de 50 éléments

— d'un tableau M$ de 25 éléments

b) Elles réservent les emplacements :

— d'un tableau C3 de 25 éléments
— d'un tableau X8 de 4 éléments
— d'un tableau Z5$ de 15 éléments

XI.9. Elles placent dans le tableau H, les valeurs 3, 6, 8 et 9 (dans cet ordre).

XI.10. Elles placent dans le tableau K1 les valeurs 1, 3, 5, 7 et 9 (dans cet ordre).

XI.11. Elles placent dans le tableau C les valeurs

1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0

XI.12. Elles placent dans le tableau Z9 les valeurs :

1, 2, 3, 4, 99, 1, 2, 3, 4, 5

XI.13.

```
180 FOR K = 1 TO 10
190 PRINT Z9 (K)
200 NEXT K
```

XI.14.

```
100 LET S = 0
110 FOR I = 1 TO 1000
120 LET S = S + V (I)
130 NEXT I
```

XI.15. a) 50 b) 22 c) 10 (par défaut)

XI.16. a) 21 b) 24 c) 25 d) 25 e) 26

XI.18. a) 1
4
9
16

b) 1
4
9
16
25
Subscript out of range in 130

Chapitre XII

XII.1. Elle demande à Basic de poursuivre l'exécution à l'un des numéros de ligne mentionnés après le GO TO, en fonction de la valeur de l'expression arithmétique placée après ON.

XII.2. a) faux b) vrai c) faux d) faux
e) vrai f) faux

XII.3. a) Si X vaut 1 on va en 100
Si X vaut 2 on va en 300
Si X vaut 3 on va en 200
Si X vaut 4 on va en 140

b) Si K + 5 vaut 1, c'est-à-dire si K vaut – 4, on va en 300
Si K + 5 vaut 2, c'est-à-dire si K vaut – 3, on va en 350
Si K + 5 vaut 3, c'est-à-dire si K vaut – 2, on va en 450

XII.4. a) Basic va en 150 b) Basic va en 150
c) Basic va en 300

XII.5. a) Basic va en 400 b) Basic va en 200

XII.6. X doit avoir sa partie entière comprise entre 1 et 4, donc les valeurs possibles sont :

$1 \leqslant X < 5$

XII.7. On doit avoir : $1 \leqslant 2 * X + 1 < 4$
donc : $0 \leqslant X < 1,5$

XII.8. Dans les deux cas, on est en dehors des valeurs permises pour X. Suivant le Basic utilisé, il y aura soit passage à l'instruction suivant le ON, soit écriture d'un message d'erreur.

XII.9. Par : 100 GO TO 200

Chapitre XIII

XIII.1. Elles s'écrivent sous la forme d'un nom de fonction suivi, entre parenthèses, d'une expression arithmétique.

XIII.2. Dans n'importe quelle expression arithmétique.

XIII.3. a) ABS (A * X + B) b) ABS (A) + ABS (B)

XIII.4. On obtiendra, au deuxième tour de boucle, le message :

Illegal function call in 130 (car X vaudra – 4)

XIII.5.

```
100 PRINT "DONNEZ LES LONGUEURS DES DEUX COTES"
110 PRINT "DE L'ANGLE DROIT D'UN TRIANGLE RECTANGLE"
120 INPUT X, Y
130 LET H = SQR (X ^2 + Y ^2)
140 PRINT "LONGUEUR DE L'HYPOTÉNUSE :" ; H
```

XIII.6. a) 8 b) 8 c) – 8

XIII.7. 9
8

XIII.8. En calculant INT (X + 0.5)

XIII.9. En calculant 10 * INT (X/10 + 0.5)

XIII.10. a) NON b) OUI

XIII.11.

```
100 INPUT A, B
110 LET E1 = COS (A + B)
120 LET E2 = COS (A) * COS (B) – SIN (A) * SIN (B)
130 PRINT E1 ; E2 ; E1 – E2
```

XIII.12. a) vrai b) faux (en général)
c) vrai (toutefois, il faut parfois éviter de lui donner la valeur 0)
d) faux (c'est une instruction)
e) vrai (en général)

XIII.13. a) ne rien faire de spécial
b) utiliser l'instruction RANDOMIZE (ou son équivalent)

Chapitre XIV

XIV.1. C'est une boucle dont le nombre de tours (non prévu à l'avance) est basé sur l'apparition (ou la disparition) d'une certaine condition.

XIV.2. Le nombre de tours est déterminé lors de l'entrée dans une boucle avec compteur. Ce n'est plus vrai dans le cas d'une boucle conditionnelle.

XIV.3. A répéter des instructions jusqu'à ce qu'une certaine condition devienne vraie.

XIV.4. A répéter des instructions tant qu'une certaine condition est vraie.

XIV.5. a) faux b) faux c) vrai

XIV.6. faux

XIV.7. faux

XIV.8. Choix — Choix multiple — Boucle avec compteur — Boucle jusqu'à — boucle tant que

Chapitre XV

XV.1. a) 2 indices b) de 1 à 4 c) de 1 à 3
d) 12

XV.2.
```
100 FOR I = 1 TO 20
110      FOR J = 1 TO 50
120           LET R (I, J) = 1
130      NEXT J
140 NEXT I
```

XV.3.
```
100 DIM T (2, 5)
110 DATA 5, 12, 8, 1, 6, 6, 9, 2, 15, 8
120 FOR I = 1 TO 2
130      FOR J = 1 TO 5
140           READ T (I, J)
150      NEXT J
160 NEXT I
```

XV.4.
```
100 DIM X$ (2, 3)
110 DATA UN, UNE, DES, LE, LA, LES
120 FOR I = 1 TO 2
130      FOR J = 1 TO 3
140           READ X$ (I, J)
150      NEXT J
160 NEXT I
```

XV.5. Première solution :

```
200 FOR I = 1 TO 2
210     PRINT TAB (5) ; X$ (I, 1) ; TAB (20) ; X$ (I, 2) ; TAB (35) ; X$ (I, 3)
220 NEXT I
```

Deuxième solution :

```
200 FOR I = 1 TO 2
210     FOR J = 1 TO 3
220         PRINT TAB (15 * (J – 1) + 5) ; X$ (I, J) ;
230     NEXT J
240     PRINT
250 NEXT I
```

XV.6. Première solution :

```
200 FOR J = 1 TO 3
210     PRINT TAB (1) ; X$ (1, J) ; TAB (10) ; X $ (2, J)
220 NEXT J
```

Deuxième solution

```
200 FOR J = 1 TO 3
210     FOR I = 1 TO 2
220         PRINT TAB (9 * (I – 1) + 1) ; X$ (I, J)
230     NEXT I
240 NEXT J
```

XV.7.

```
100 DIM T (3, 4)
110 LET K = 5
120 FOR I = 1 TO 3
130     FOR J = 1 TO 4
140         LET T (I, J) = K
150         LET K = K + 5
160     NEXT J
170 NEXT I
```

Chapitre XVI

XVI.1. Vrai

XVI.2. faux (la deuxième instruction est d'ailleurs incorrecte)

XVI.3. C'est une variable qui est utilisée dans la définition d'une fonction (instruction DEF FN...) et qui n'a aucune existence réelle, en dehors de cette définition.

XVI.4. – 1 ^^1 (^ désignant le caractère blanc)

XVI.5. ^0 ^^10 ^^3 (^ désignant le caractère blanc)

XVI.6. a) vrai b) vrai

XVI.7. En 100, Y est une variable muette, tandis qu'en 110 ou en 130, Y est un argument effectif.

XVI.8. Faux. On peut y trouver d'autres variables.

XVI.9. 3
4

XVI.10. 12
12

XVI.11. La fonction FNC est utilisée (en 110). avant d'être définie (en 130). L'exécution du programme produira un message du type :

Undefined user function in 110

XVI.12. La définition de FNL (en 100) comporte une erreur de syntaxe (il manque le symbole * entre 2 et EXP). L'exécution de ce programme fournira un message du type :

Syntax error in 120

Chapitre XVII

XVII.1. C'est un ensemble d'instructions, parmi lesquelles on rencontre au moins une instruction RETURN, que l'on utilise autant de fois que nécessaire par l'intermédiaire d'une instruction GOSUB.

XVII.2. A se débrancher à un sous-programme et à conserver le numéro de l'instruction suivante afin d'y revenir lors de la rencontre d'un RETURN.

XVII.3. A revenir à l'instruction suivant le dernier GOSUB exécuté.

XVII.4. Vrai — mais une erreur se produira lors de la rencontre d'un RETURN.

XVII.5. L'exécution du programme commence en 100. Lorsque Basic rencontre l'instruction 130 RETURN, il ne sait pas où revenir, puisqu'il n'est pas entré dans le sous-programme par GOSUB. Ici, il aurait fallu, soit «sauter», les instructions 100 à 130 par un GO TO approprié, soit placer le sous-programme à la fin.

XVII.6. Vrai

XVII.7. A se débrancher à l'un des sous-programmes mentionnés après le GOSUB, en fonction de la valeur de l'expression arithmétique suivant ON.

XVII.8. Toutes deux réalisent un branchement à un numéro de ligne dépendant d'une expression arithmétique. Mais, en outre, ON GOSUB demande à Basic de conserver le numéro de l'instruction suivante, ce qui n'est pas le cas pour ON GO TO.

Chapitre XVIII

XVIII.1. non

XVIII.2. a) 3 b) 7 c) 5

XVIII.3. C'est le nombre de valeurs différentes que peut prendre chaque position.

XVIII.4. $10^3 = 1000$ informations différentes.

XVIII.5. $3^2 = 9$ informations différentes qui sont :

aa ab ac ba bb bc ca cb cc

XVIII.6. C'est une position d'un code à 2 moments.

XVIII.7. a) 2 b) $2^2 = 4$ c) $2^4 = 16$
d) $2^8 = 256$ e) $2^{16} = 65536$

XVIII.8. deux types : numérique et chaîne de caractères

XVIII.9. le code ASCII

XVIII.10. * la représentation approchée
* les limitations sur les valeurs absolues des nombres représentables.

XVIII.11. le type double précision permet de représenter les nombres de manière plus précise que le type réel. En général, les limites sur les valeurs représentables sont les mêmes dans les deux cas.

XVIII.12. Le type réel permet de représenter des nombres très grands en valeur absolue (jusqu'à 10^{38}) ou très petits (10^{-38}) avec une certaine erreur. Le type entier permet de représenter des nombres dont la valeur absolue ne dépasse pas 32767, mais sans erreur.

XVIII.13. C'est une instruction qui permet de définir le type de plusieurs variables, en fonction de la première lettre de leur nom.

XVIII.14. Elle consiste à définir le type d'une variable en ajoutant un caractère particulier à la fin de son nom.

XVIII.15.

K	entière (déclaration implicite)
IN$	chaîne (explicite)
CARAC	réelle double précision (implicite)
Y7%	entière (explicite)
AZUR!	réelle simple précision (explicite)
POLE	entière (implicite)
Y$	chaîne (explicite)

XVIII.16. Il a lieu lorsque la valeur absolue du résultat d'un calcul sort des limites possibles pour le type correspondant.

XVIII.17. Au bout d'un certain temps, on obtiendra :

Overflow in 110

XVIII.18. La plupart du temps, il remplace le résultat par la plus grande valeur représentable et il poursuit l'exécution.

XVIII.19. Il écrit un message :

Division by zéro in nnn

Souvent, le résultat est remplacé par la plus grande valeur représentable, et l'exécution se poursuit.

Chapitre XIX

XIX.1. Elle fournit la longueur d'une chaîne de caractères

XIX.2. a) 5 b) 12 c) 0

XIX.3. LEFT$ extrait d'une chaîne, une sous-chaîne formée des premiers caractères de gauche.

RIGHT$ extrait d'une chaîne, une sous-chaîne formée des derniers caractères de droite.

XIX.4.

a)
```
MR
MOUR
MONEUR
MONSIEUR
MONSISIEUR
MONSIENSIEUR
MONSIEUONSIEUR
MONSIEURMONSIEUR
```

b)
```
RR
REER
REVVER
REVEEVER
REVERREVER
```

XIX.5.

```
100 INPUT C$
110 FOR I = 1 TO LEN (C$)
120 PRINT MID$ (C$, I, 1)
130 NEXT I
```

XIX.6. ELLE (à moins que votre Basic ne vous signale une erreur en 110 ou en 120).

XIX.7. C'est l'opération qui consiste à créer une chaîne en en mettant deux autres bout à bout.

XIX.8. MIEUX VAUT RIRE QUE PLEURER

XIX.9. STRING$ crée une chaîne formée de la répétition d'un caractère donné.

SPACE$ fait la même chose avec le caractère blanc.

XIX.10. aucune

XIX.11. a) B b) B

XIX.12. a) F b) O

INDEX ALPHABÉTIQUE

P

R

S

T

U

V

Imprimé en France. — JOUVE, 18, rue Saint-Denis, 75001 PARIS
N° 10907. N° d'Editeur : 3879. — Dépôt légal : Janvier 1983

HIRSCH - **Le Basic facile par méthode progressive** - 288 p., 1982 (coll. Pratique de l'Informatique)

LAMOITIER - **Le langage Basic et la nouvelle norme** - 232 p., 1982

LARRÉCHÉ - **Le Basic**, une introduction à la programmation - 120 p., 1981

LONCHAMP - **Basic.** Construction méthodique des programmes - 136 p., 1983 (coll. Pratique de l'Informatique)

NEVISON - **L'art de bien programmer en Basic.** Le petit livre du style - 128 p., 1982 (coll. Pratique de l'Informatique)

PARKER & SILBEY - **Le Basic en gestion** - 272 p., 1981 (coll. Pratique de l'Informatique)

QUANEAUX - **Exercices de gestion en Basic** - 136 p., 1983 (coll. Pratique de l'Informatique)

DE ROSSI - **Apprentissage rapide du Basic** - 216 p., 1982 (coll. Pratique de l'Informatique)

Collection Micro-ordinateurs

CANAL - **Parler L.S.E. et apprendre à l'utiliser** - 160 p., 1982

HERNANDEZ - **Pascal par l'exemple** - 152 p. + 4 p. couleur, 1982

NOLLET - **La conduite du ZX 81** - 128 p., 1983

PELLIER - **La conduite du TRS 80** - 120 p., 1982

SCHOMBERG - **Le Basic universel** - 128 p., 1982

SCHOMBERG - **Micro-ordinateurs : comment ça marche ?** - 96 p., 1982



8614

EYROLLES

APPRENDRE A PROGRAMMER EN **BASIC**

C. Delannoy



AMSTRAD

CPC

MÉMOIRE ÉCRITE

https://acpc.me/